Daniel Geiger, BSc

# Integration of Matlab-Simulink/Software-In-The-Loop Models in VECTO

**MASTER THESIS**

to achieve the university degree of

Diplom-Ingenieur

submitted to

**Graz University of Technology**

Supervisor

Dipl.-Ing. Dr.techn. Georg Macher BSc MBA
Institute of Technical Informatics

Technical Supervisor

Dipl.-Ing. Dr.techn. Markus Quaritsch
Institute of Thermodynamics and Sustainable Propulsion Systems

Graz, April 2022

# Preface

Due to my experience with C# and engines during my student job, the topic of this thesis looked very interesting to me and therefore I decided to learn more about it. The climate change is a problem that everyone will need to face in our generation and will impact all future generations as well. Every tiny step is needed to combat this problem.

I would like to thank my partner in life for the support and motivation to continue and finish my studies. As well as the support of my family and future family in-law. Especially I would like to thank everyone who read over my thesis and made suggestions.

A special thanks also to my technical adviser Markus for all the support and answers to my questions or when I was stuck at a problem. I also want to thank my supervisor Georg for the support, as well as Michael for finding this thesis topic and the support.

# Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

*Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.*

Graz, 5.4.2022 Daniel Geiger

# Abstract

The simulation software VECTO is responsible for the certification process of heavy-duty vehicles to calculate the $CO_2$ emissions in the European Union. The manufactures have their own strategies when to shift gears depending on many parameters, but VECTO has only generic shift strategies implemented. The manufacturer shift strategies are implemented as Matlab Simulink models. This thesis researches the possible technologies on how to connect VECTO, which is written in C#, to a Matlab Simulink model. The decision was made on the technology to export the model to C code and to compile it into a shared library DLL file. The FMI standard will be added to the shared library as well in order to run the model from C# code with additional possibilities. In the first part the generic shift strategy of VECTO was implemented as a Matlab Simulink model and tested with these technologies. In the second part a hybrid strategy of an OEM will be used to test the technology. In the Evaluation chapter it is concluded that the technologies work very well and from a technical aspect, the connection between VECTO and a Matlab Simulink model is possible with minor development effort. However, there are some questions left open as an external model can influence the simulation drastically.

# 1 Kurzfassung

Die Simulationssoftware VECTO ist verantwortlich für die Zertifizierung von schweren Nutzfahrzeugen durch die Berechnung von den $CO_2$ Emissionen in der Europäischen Union. Die Hersteller haben eigene Strategien um den Gang basierend auf bestimmten Parametern zu wechseln, jedoch hat VECTO nur generische Schaltstrategien implementiert. Die Schaltstrategien von den Herstellern sind als Modelle in Matlab Simulink implementiert. Das Ziel dieser Arbeit ist esmögliche Technologien herauszufinden wie man VECTO, welches in C# programmiert wurde, mit einem Modell in Matlab Simulink verbindet. Es wurde der Entschluss gefasst, das Modell in C code zu exportieren und dann in eine gemeinsame Bibliothek, eine DLL, zu kompilieren. Außerdem wurde der FMI Standard hinzugefügt zu dieser Bibliothek, um mehr Möglichkeiten zu haben sie in C# auszuführen. Im ersten Teil dieser Arbeit wurde die generische Schaltstrategie von VECTO in Matlab Simulink Model implementiert um diese Technologie zu testen. Im zweiten Teil dieser Arbeit wurde eine Hybridstrategie als Matlab Simulink Model von einem Hersteller verwendet. In der Evaluierung wurde zusammengefasst, dass die Technologie sehr gut funktioniert, und dass von dem technischen Standpunkt die Verbindung zwischen VECTO und einem Matlab Simulink Model mit minimalen Entwicklungsaufwand möglich ist. Jedoch gibt es auch noch einige offene Themen, wie zum Beispiel, dass diese externen Modelle die Simulation drastisch beeinflussen können.

# Contents

# Formula Symbols, Indices and Abbreviations

## Indices and Abbreviations

| | |
|---|---|
| ADAS | Advanced Driver Assistance System |
| AI | Artifical Intelligence |
| AMT | Automated Manual Transmission |
| API | Application Programming Interface |
| AT | Automated Transmission |
| BCVTB | Building Controls Virtual Test Bed |
| CFD | Computational Fluid Dynamics |
| CIL | Component-in-the-Loop |
| COM | Component Object Model |
| CST | Constant Speed Cycle |
| DLL | Dynamic Link Library |
| EC | European Commission |
| FMI | Functional Mockup Interface |
| FMU | Functional Mockup Unit |
| GHG | Greenhouse Gas |
| GUID | Global Unique Identifier |
| HTTP | Hypertext Transfer Protocol |
| HDV | Heavy-Duty Vehicles |
| HIL | Hardware-in-the-Loop |
| ITnA | Institute of Thermodynamics and Sustainable Propulsion Systems |
| ICE | Internal Combustion Engine |
| MEX | Matlab Executable |
| MIL | Model-in-the-Loop |
| OEM | Original Equipment Manufacturer |
| PCC | Predictive Cruise Control |
| RCP | Rapid-Control-Prototyping |
| EC | Rechargeable Electric Energy Storage System |
| RRC | Rolling Resistance Coefficient |
| SIL | Software-in-the-Loop |
| SOAP | Simple Object Access Protocol |
| SOC | State of Charge |
| TLC | Target Language Compile |
| VECTO | Vehicle Energy Consumption Calculation Tool |
| VTP | Verification Testing Procedure |
| WSDL | Web Services Description Language |
| W/EHR | Waste/Exhaust Heat Recovery |
| WHTC | World Harmonized Transient Cycle |

# 2 Introduction

Emissions reduction is a growing topic that everyone on our planet is aware by now due to the climate crisis. In order to reduce the rising temperatures on earth and to stop the green house effect, actions need to be taken. In the future the emissions of every sector needs to be reduced. One of those is the transportation sector, as since the 1990s the emissions in this sector have been rising steadily.

The EU set targets in 2007 for the year 2020 to cut back emissions in different sectors and legislated these targets in 2009. The targets were to reduce the greenhouse gas emissions by 20% from the levels of 1990, to use 20% renewable energy sources and to improve the energy efficiency by 20%, this is the so called 20/20/20 plan. Part of these targets included the transport sector and heavy-duty vehicles as they are responsible for a large share of the emissions in the transport sector.

In the EU the improving technologies to be more energy efficient and to reduce emissions were not sufficient and the number of cars on the streets were rising as well. Therefore, the EU set new targets to force manufacturers on improving their vehicle's energy efficiency even more, in order to reduce emissions and to maintain a leadership role in technologies in Europe.

In order to certificate new vehicles with the reduced emissions a process was needed for testing the vehicles to assess the emissions of each vehicle. Heavy-duty vehicles are available in a huge variety and testing each vehicle combination was not feasible. Therefore, the emissions needed to be calculated in a simulation based on available data about the components of the vehicle. VECTO was developed by the Institute for Internal Combustion Engines and Thermodynamics at the Graz University of Technology for the European Commission. VECTO simulates the driving of vehicles for a certain path and calculates their $CO_2$ emissions.

Part of improving the energy efficiency of heavy-duty vehicles are new algorithms, strategies and assistance systems. These improvements need to be included in the VECTO simulation, in order to declare lower $CO_2$ emissions for a vehicle. The manufacturers have models of these systems that are proprietary and cannot be made public.

The aim of this thesis was to look into how such a model of a manufacturer can be included in the simulation of the vehicles in VECTO, while still maintaining the intel-

lectual property of the manufacturers models and to let them use their own strategies to improve their $CO_2$ certification emissions.

## 2.1 Motivation

The climate crisis is affecting every one, maybe not yet significantly, but it will affect all future generations. Solutions to tackle the problems of $CO_2$ emissions are needed and to be found in the next few years in order to reduce the rising temperature of earth. Technology can help in these aspects, but with improving technology complexity increases too.

Software development in the automotive sector is getting more complex with the addition of more electronics, sensors and optimized component strategies in the vehicle. Additional systems such as driver assistance systems increase the complexity as well. With all of these systems the safety requirement standards need to be passed as well. Not only increased the complexity of the software due these new systems, but shorter development times for new vehicles, increasing competition and increasing safety requirements are the challenges that need to be taken into consideration. In order to solve these development difficulties Sivakumar P. et al. researched in their paper [1] how to use a model based design approach.

The use of system designs support such as "Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), Hardware-in-the-Loop (HIL), Rapid-Control-Prototyping (RCP), or Component-in-the-Loop (CIL)" as described in their paper help to support the various development phases. "Verification, validation and testing at the end of the process" are the main differences between the model based approach and a traditional development process. An additional advantage is the option to autogenerate code of a model which is much better than hand-written code, as it is less prone to developer mistakes. [1]

Due to the use of such model based development the models will be optimized for a specific vehicle and its parameters as well. The model will not only be optimized with the vehicle parameters, but a specific route as well. The individual strategies for switching gears or for hybrid vehicles can improve the vehicle emissions and fuel efficiency drastically. The OEMs want to use these models for the calculation of the $CO_2$ emissions in VECTO to get a reduction of emissions for the certifications as well. VECTO only uses a generic strategy in where it does not matter which vehicle it is, whether it is a 10 or 40 ton vehicle, or if it has 4 or 12 gears. For this reason this master thesis will research if and how it is possible to connect VECTO with such a vehicle model during the VECTO simulation.

# 3 Background & Related Work

## 3.1 VECTO

The tool "Vehicle Energy Consumption Calculation Tool" (VECTO)[1] was developed for the European Commission (EC) by the Institute of Thermodynamics and Sustainable Propulsion Systems (ITnA) at the Graz University of Technology. It calculates the fuel consumption and $CO_2$ emissions of heavy-duty vehicles (HDV). All the main components of an HDV are simulated in VECTO. This is done with parameters to model each component and the whole vehicle. VECTO is mandatory to be used for all new HDV in the EU as part of the $CO_2$ certification procedure since 1. January 2019. [7]

### 3.1.1 History

Greenhouse Gas (GHG) emissions in the transport sector have been on the rise since the 1990s. As of April 2009 regulations to reduce the emissions of passenger cars have already been approved. Additional plans for light goods vehicles were also being approved. However, the EC wanted to reduce the rising emissions from heavy-duty vehicles too, as they are responsible for 26% of all $CO_2$ emissions of road transport in the EU. [4]

According to the report by Hausberger S. et al. [5], vans and heavy-duty vehicles have a wide variety of applications as opposed to passenger cars. They vary in size, weight and components used, which results in many combinations. The EC wanted a test procedure for certification of new heavy-duty vehicles, to limit the energy consumption of the components and therefore reduce the emissions of the vehicle. Testing all different combinations of a vehicle on a test bed or on the road would not be feasible. It would be very costly and some vehicle configurations are only sold in limited numbers. The procedure for passenger cars and light commercial vehicles to measure $CO_2$ levels are chassis dyno measurements, a test bed where the vehicles can be tested. [3] The standard procedure for the assessment of regulated emission components of heavy-duty vehicles is the engine on a test bed. Hence, this would only cover the efficiency of

[1] https://ec.europa.eu/clima/eu-action/transport-emissions/road-transport-reducing-co2-emissions-vehicles/vehicle-energy-consumption-calculation-tool-vecto_en

the engine and not all the other components of a heavy-duty vehicle. Therefore, a new testing procedure was needed in order to cover all the components and their efficiency to get the overall emissions of the vehicle.

The proposed test procedure is based on testing each component, to get the efficiency and other parameters of the components. The data of these can then be used in a simulation tool to calculate the overall emissions of the vehicle. Additionally, resistance forces and component losses are included in the calculation. This procedure was described in the final report by Hausberger S. et al. [5]

This test procedure and the simulation tool were further developed in the report by Luz R. et al. [6]. A proof-of-concept phase was conducted which showed accurate results for the component test procedures and the simulation tool. One of the deliveries of this report were the descriptions of the component tests that were going to be used as input for the regulatory text. Another delivery was the VECTO software, which uses the component data to calculate the HDV $CO_2$ emissions. The generic data of some components are used for different vehicle classes that are completed in this report such as auxiliaries or mass of standard bodies and trailers. With a constant speed test procedure the aerodynamic drag is measured and can be evaluated with the VECTO-CSE software which needed a major update due to further development of the aerodynamic drag test procedure. The software helps to evaluate the aerodynamic drag test data. The "draft certification report" describes the possible certification process and is included in the report as well. The description of the component testing methods, of the evaluation and of the simulation of these methods and how the generic data is used in VECTO for the certification is included in the report as well. [6]

In the report by Rexeis M. et al.,[7] the certification procedure was finalized for a legislative procedure. Additional tasks of the report were the development of the methodologies for component testing, the software for said methodologies, more detailed descriptions of the input and generic data for the software, assistance for users and discussion with stakeholders. The VECTO software was at a suitable stage to be used for the certification process, including a set of generic data of all the necessary components for different vehicle groups. In the report further vehicle groups were described that have not yet been covered, such as buses, coaches, intermediate-sized vehicles et cetera. It was described the methodologies and needed steps to include these vehicle groups. The vehicle groups planned to be included in the $CO_2$ certification process cover about 98% of the $CO_2$ emissions of HDV in Europe. At the end of the report, followup activities were listed and described for SR7 (the current project called Service Request 7) to be finalized. Identified medium-term goals were the inclusion of Hybrid Electric Vehicles and the discussion of advanced engine technologies such as Waste Heat Recovery. A long-term goal to elaborate is the option for Original Equipment Manufacturers (OEMs) to use their own control strategies in VECTO. This process would be complex and time-consuming in the range of several years. [7]

Starting with 1. January 2019 VECTO was mandatory to be used in the European Union for new trucks that fall under the categories in the certification process. A few shortcomings that were addressed in the report [9] were fuel efficiency technologies, generic gear shift algorithms for automated manual transmission (AMT) and automated transmissions (AT), Predictive Cruise Control (PCC) systems, Waste/Exhaust Heat Recovery (W/EHR) systems, gas and dual-fuelled engines and therefore VECTO had to be updated with the mentioned additions. The new gear shift strategies called "EffShift AMT" and "EffShift AT" were introduced to handle overdrive transmissions and all HDV configurations. Advanced Driver Assistance Systems such as Engine Stop-Start, Eco-Roll and Predictive Cruise Control were implemented. Gaseous fuels with lower carbon content than diesel fuel can reduce $CO_2$ emissions of vehicles. Dual-fuel engines use both diesel and gas fuels simultaneously and will be coming to market soon. Advanced algorithms and technologies such as gear shifting, Advanced Driver Assistance Systems (ADAS), hybrid vehicles and others can improve the fuel efficiency as well. This possibility to let the OEMs use their own strategies is evaluated and could be implemented with one out of three options. The first option is a software-in-the-loop (SIL) controller OEMs can utilize to connect their own strategies to VECTO. The second option is a Verification Testing Procedure (VTP) which uses a simple fuel map with vehicle speed and wheel power as the abscissa and ordinate coordinates, that replaces the drivetrain in the simulation. The last and third option is the use of artificial intelligence (AI) tools. The first option will be reviewed in the LONGRUN Project [2]. Option two would be a simple approach to certify complex new $CO_2$ technologies. Option three could not be tested due to no available data. [9]

Hybrid Heavy-Duty Vehicles were still missing in VECTO and with that, a rising share of HDV in the EU were not included in the $CO_2$ emission and fuel consumption certification process. In the report by Silberholz G. et al.[10] the current situation of Hybrid HDV was assessed and solutions provided to include these hybrid vehicles in the certification process and to extend VECTO to support them. As the inclusion of hybrid vehicles in VECTO would be a complex process, other alternatives were also investigated that would still match the results of VECTO. In the end using VECTO as the simulation tool for hybrid HD vehicles was the preferred method for the authors and other stakeholders alike. Other vehicle types such as mild hybrids, plugin Hybrid Electric Vehicles (PHEVs) and pure battery electric cars need to be included in the extension of VECTO too. [10]

Other HDV that are not yet included in VECTO and the certification process are trailers and vehicles with superstructures. Superstructures are box types, curtain sides or flat beds and more. The tasks of the report by Luján C. et al. [12] were to analyze the current market of trailers, semi-trailers and bodies, investigate methods to create and handle input data for VECTO with trailers and bodies and to evaluate the results

[2] `https://cordis.europa.eu/project/id/874972`

of the methods for the input data of VECTO in regard to bodies on rigid HVD and trailers. Additionally, a task of the study was to analyze aerodynamic benefits to reduce the drag of trailers for HDV. It was concluded that the variety of regulations and methodologies needs harmonization to achieve a global fuel efficiency regulation. With relatively low effort input data and lookup-tables can be created for VECTO for the certification of add-on components such as boat tails. Incentivizing the use of aerodynamic optimizations can be done through using Computational Fluid Dynamics (CFD) methods instead of component tests. In the end the report [12] concluded four different methods can be used to provide reliable results. The first method is to ignore the air drag for trailers and bodies. The second method uses look-up tables and through a deep analysis provides the best solution for all stakeholders. The third option are speed tests and the fourth one is the CFD method. [12]

### 3.1.2 Components & Input

The main fuel consumption components that need a test procedure are the engine, transmission and other torque transmitting components like axle, air drag and tires. The results of these component tests are used as input for the VECTO simulation.

#### Engine

For the engine test, the engine needs to run 5 test runs to get all the necessary data, but two of them can be skipped when the engine is part of the same engine $CO_2$-family. The resulting data of these test runs is the input for the VECTO Engine pre-processing tool. The required files are the fuel consumption map, the full load curve of $CO_2$-parent engine, the full load curve of the actual engine and the motoring curve of the $CO_2$-parent engine. Additional parameters such as the idle speed, rated power, fuel type, transient correction factors obtained from the World Harmonized Transient Cycle (WHTC) and more are required. The tool processes the data to be used as input for the VECTO simulation. [7]

#### Air Drag

Air drag data in VECTO is gathered by the Constant Speed Test (CST) procedure. In the test the following signals are measured: air flow, total driving resistance force and vehicle position for the "measurement sections". With all this measurements, the air drag can be calculated which is described by Rexeis et al. [7, page 29] as: "...the aerodynamic drag of the vehicle given by the product of air drag coefficient (Cd) with the frontal area (Afr) of the vehicle at zero-wind conditions (yaw angle $\beta$=0)". With the Air drag pre-processing tool the measured data can be evaluated and the corresponding VECTO input files are created. [7]

#### Transmission

Transmission data consist of the gear ratios, loss maps, gear dependent input torque limits and input speed limits. There are three measurement options that can be used to measure the needed values, which differ from each other in measurement effort and accuracy. Typically, when using methods with less measurement efforts the losses are higher. The option two is balanced between effort and accuracy, as the torque dependent losses are calculated by interpolation as opposed to option one where it is calculated. Option three is the most complex method out of the three methods as the total torque loss is measured. There is no common tool for evaluating of the test procedure data and creating the input data for VECTO, this has to be done by the OEMs according to the procedure defined in the EU Regulation. [7]

#### Axle

The required axle torque losses can be acquired by two options. The first is to use generic values which depend on the axle configuration. The second option is a physical test for one axle of an axle family. As for the Transmission, there is no common tool to evaluate the data. [7]

#### Tires

The rolling resistance coefficient (RRC) of HDV tires need to be tested on a test drum. These RRC values are used as the input for VECTO. Road tests are not necessary as some tolerances for the class labels of tires, which are rated from "A" to "G", are applied. [7]

#### Auxiliaries

Auxiliaries provide functions to the vehicle that do not relate to moving it forward. Some auxiliaries are already included in the fuel map of the engine such as the engine oil pump, coolant pump, fuel delivery pump and fuel high pressure pump. Other auxiliaries that need to be included in the fuel consumption of the vehicle individually are the engine cooling fan, alternator, air compressor, steering pump or the A/C compressor. For simplicity the power consumption of the auxiliaries are added as a constant power level to the combustion engine. [7]

### 3.1.3 Calculation

There are two methods to calculate a longitudinal dynamics model, the forward-calculating and the backward-calculating method. The forward-calculating method

calculates from the engine to the wheel, depending on the acceleration of the vehicle. The backward-calculating model calculates from the wheel back to the engine. Both of them need the same data, but the advantage of the backward-calculating model is that no differential equations need to be solved as with the forward-calculating method. The driving-cycle sets the driving action, speed and slope. With this information the state of the wheel is determined and calculated backwards to the engine, therefore the engine state can be calculated in the end. [7]

### 3.1.4 Software Architecture

VECTO consists of the Input Module, Simulation Module, Output Module and the Graphical User Interface (GUI). The Input Module is responsible for reading in all the different parameters and model data in their respective data file formats. All the data files of VECTO are in the data formats JSON, CSV or XML. The Simulation Module consists of the Simulator Factory and the Simulation Core. The Simulator Factory is responsible for creating the required model parameters and powertrain for the simulation. A single vehicle can be simulated in different driving cycles, which can be run in parallel. The Simulator Core contains all the components of the vehicle, each has an interface and their respective implementation with their input, model parameters and output. Model parameters are various maps or generic values. The output of one component is the input of the next component, in this manner a modular approach is achieved and components can be exchanged, depending on their compatibility. The generic component structure can be seen in Figure 3.1. The flow of direction is from the wheels component towards the engine component, the input of a component is therefore in the direction of the wheels and the output of a component is in the direction of the engine. The naming of the input and output parameters are in the direction of the power flow while driving normally, which means propulsion forwards. [7]

Results of the simulation driving cycles are collected in the output module, which is responsible for generating the reports with all the necessary descriptions, parameters and performance results of the vehicle. The generated reports can be created in different data formats. [7]

### 3.1.5 Simulation

In a driving cycle the target speed has to be followed over the distance. Every vehicle therefore drives the same distance and needs to follow the set target speed as exact as possible. The distance in a single simulation step is varying depending on the speed of the vehicle, but one steps takes about 0.5 seconds. The transition from the distance to the time in a simulation step happens in the driver component. In VECTO a request for a valid operating point starts from the simulator over the driving cycle, driver and

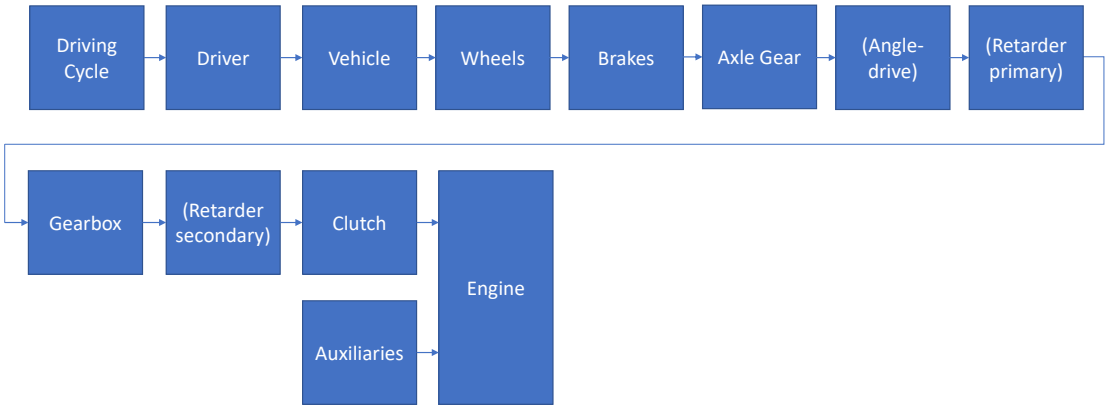


**Figure 3.1:** Structure of the powertrain components in VECTO [Redrawn from Figure 6 of [7]]

vehicle, wheels downwards the component structure to the engine. All components get the output of the last component as their input. At the end of the components chain the engine returns the results back upwards the components with an error or success message and each component adds their results. When a request results in an error it is repeated with changed parameters until a success is achieved. After a successful request of the whole component structure, the simulation step is committed for each component and their respective data is sent to the output module. Afterwards the component states are updates and the next simulation step can start. [7]

Depending on the response of a request, the driving strategy can reduce the acceleration, when the torque demand is too high for the engine or when the torque demand is too low, the brakes can be used. To search for such a valid operating point, a "dry-run" request is issued. A dry run is handled differently in some components, thus only the necessary data is calculated to search for a valid-operating point. Is a valid point found, the normal request is issued again with the found acceleration or breaking force. [7]

### 3.1.6 VECTO Outputs

VECTO outputs different files depending on the Declaration Mode and Engineering Mode. In the Declaration Mode the Manufacturer Report and the Customer Information Report are created. In the Engineering mode the summary data file and the modal results file are created, both are optional to be created in the Declaration Mode too.

The Manufacturer Report file and Customer Information Report include the vehicle description, its parameters, performance, fuel consumption and driving cycles. The modal results show the simulation details of every simulation step and data such as acceleration, speed, engine torque, engine speed, et cetera. The summary data file shows the fuel consumption and other metrics such as number of gear shifts, percent in a certain gear, average vehicle speed, et cetera. [7]

### 3.1.7 Hybrid VECTO

Hybrid Heavy Duty Vehicles are not yet supported in the current official version of VECTO 3.3.9.2175, but in the development version of VECTO Hybrid vehicles are already supported. For this master thesis the development version of VECTO with the support of hybrid vehicles was used in the second phase of the implementation part that can be seen in Section 6.2.

#### Hybrid Vehicle Types

There are different types of hybrid vehicles depending on the position of the electric motor in the powertrain as displayed in Figure 3.2. The first type of position is called P0, where the engine is directly connected before the electric motor. P1 is the second type where the electric motor is directly connected after the engine. P2 is the type where the electric motor is between the gearbox and the clutch. With the type P3 the electric motor is after the gearbox and in the type P4 the electric motor is directly at the rear axle.

There are also pure battery electric vehicle types included in the hybrid version of VECTO. Three different types of battery electric vehicles exist, which are all supported. The first is called the E2, where the electric motor is directly connected to the gearbox and the second is called E3, where the electric motor is directly connected to the axle input shaft. The last type is called the E4, where the electric motor is directly at the wheels.

#### Hybrid Controller & Hybrid Strategy

In order to support hybrid HDV in VECTO, additional components were added. Components for the Electric Motor, the Rechargeable Electric Energy Storage System (REESS) and a Hybrid Controller which contains the Electric Motor Controller and Hybrid Strategy had to be added as can be seen in Figure 3.3. The Electric Motor which is between the Gearbox and Clutch in the P2 Hybrid Type is connected to the REESS and the Hybrid Controller. The Hybrid Controller is between the wheels and

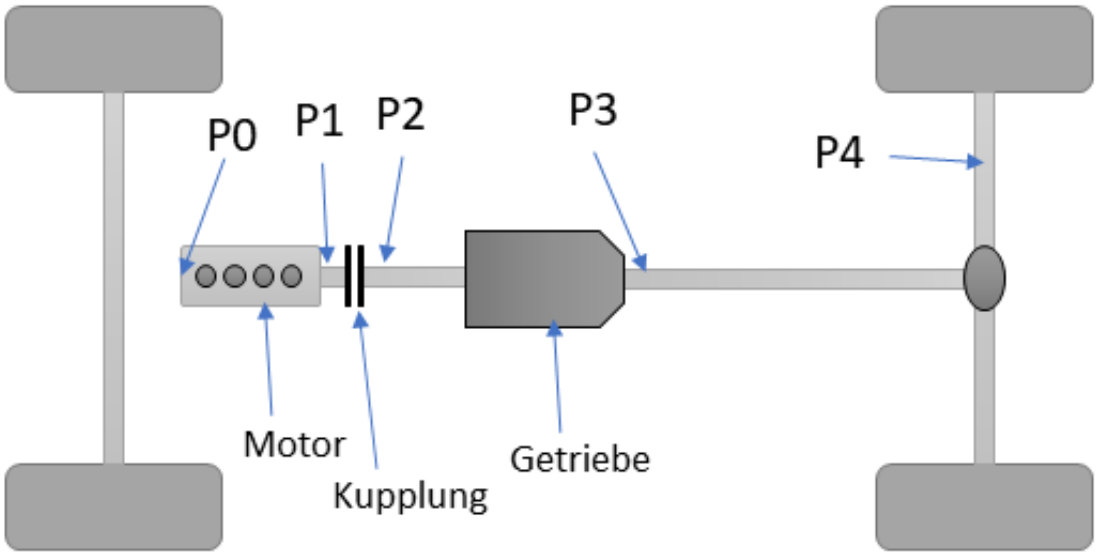


**Figure 3.2:** Hybrid vehicle types supported in VECTO [Redrawn from Figure 2 of [11]]

the brakes in the powertrain and is also connected to the gearbox, as it has to decide when it is time to shift too.

The Hybrid Controller sends a request to the Hybrid Strategy with all the necessary data. The Hybrid Strategy decides how much torque the electric motor and how much the ICE should provide. All the different configurations are tested based on when the ICE should be turned off and the electric motor should do the full work or the electric motor is off and the ICE does all the work. In order to decide on all of these possibilities and whether it should be up- or downshifted the strategy calculates a lot of Dry-Runs on the Test Powertrain which is a near replication of the original powertrain but simplified. This Dry-Run only calculates the necessary result values to compare the configurations. The vehicle is modeled in VECTO with different driving actions, which are Accelerate, Coast, Roll, Break or Halt action. Depending on the driving action it will decide on different configurations of the amount of torque the electric motor and the ICE should do and a Dry-Run is run. Additional information such as the battery state of charge (SoC) are needed in the Hybrid Strategy too. The best result will then be taken according to some rules and returned to the Hybrid Controller.

According to the result of the Hybrid Strategy the Hybrid Controller will then trigger a gearshift if necessary and the next component in the powertrain will get a request. The electric motor will, according to the strategy, apply the decided torque or whether it should propel or recuperate and the ICE will apply the rest of the torque or be turned off, depending on the strategy decisions.

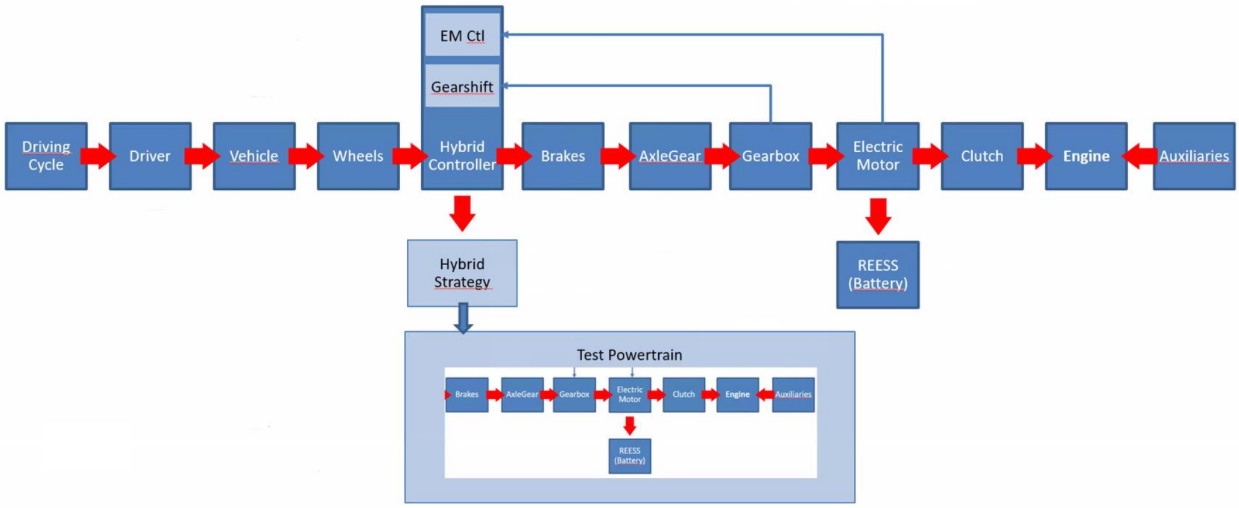


**Figure 3.3:** Structure of the powertrain components in Hybrid VECTO with P2 electric motor

## 3.2 Matlab Simulink

Matlab is a software by the company Mathworks which was founded in 1984. Matlab exists since then with the version 1.0 and at the time of this writing the newest version is R2021a. Matlab is an environment for computing, programming and mathematical purposes. Data can be analyzed and computed, algorithms developed and many more features are available. There are currently 119 different Toolboxes available, which are extensions to the base Matlab. Each Toolbox has its own purpose to extend Matlab. Such Toolboxes for example are the Aerospace Toolbox, Financial Toolbox, Lidar Toolbox or Statistics and Machine Learning Toolbox. [13]

Matlab is a programming language focused on math and science. By entering commands matrices, arrays can be created calculated or plotted. Data can be imported and exported and the scripts with all the commands can be saved and shared.[13]

### Simulink

Simulink is a Toolbox on top of Matlab and one of the biggest extensions to Matlab. Simulink is a model environment, where models with inputs and outputs can be created. It consists of block diagrams in an environment where blocks of different types such as an input, calculation, plot or export block can be placed and connected with each other. In Simulink nearly any real world application can be modelled such as a physical model, signal processing model or automotive/aerospace model.

## 3.3 Related Work

The coupling of multiple software applications in order to do a simulation is called Co-simulation. The survey and following report conducted by Gomes C. et al. [14] describes the aspects of Co-Simulation. Development of complex systems, that involves multiple groups of people and integrates hardware components, is very challenging. In order to improve the quality early testing of the partial solutions of each group is necessary. Developing models and using simulations can reduce the development efforts. The survey investigates the main challenges for the co-simulation of these models. From the work it was concluded that the main challenges are: "semantic adaptation, modular coupling, stability and accuracy, and finding a standard for hybrid co-simulation" [14].

Co-simulation is a general approach, but a master is necessary to coordinate the simulation between every model with the usage of FMI as described in the paper by Bastian J. et al. [14]. The Functional Mockup-Up Interface (FMI) provides a standardized interface to exchange data and covers algorithmic issues. Without such a standard for the different simulation tools co-simulation would lead to stability and accuracy problems. There are two types for the computation of steps. Using the Jacobi type is when each subsystem makes an assumption for its input and its output is calculated and can therefore run in parallel. After the step, the output is synchronized and the exchanged between the subsystems. The second type is called Gauß-Seidel where each slave performs a calculation after another in a sequential order. This was used in the first implementation of the paper. A disadvantage of the second type is that slaves depend on the output of each other. It is concluded in the paper that the FMI standard should be used more widely as it replaces the need for simulation tools so that only point-to-point interfaces have to be implemented. A master algorithm helps in such Co-Simulation approaches.

For Co-Simulation many different programs exist. One of the most prominent simulation programs is Matlab Simulink. Matlab can be used in a variety of ways. Extending Matlab Simulink or exporting a Matlab Simulink model in order to communicate with other software is often required in a Co-Simulation setup. Reasons for this are very different, depending on the goals to be achieved. One of the goals can be rapid-prototyping, exporting a model or extending one when it is needed to use the model outside of Matlab Simulink. Another reason can be if it is used for software-in-the-loop or hardware-in-the-loop simulations, where external software or hardware needs to communicate with Matlab Simulink or the exported model.

In order to let Matlab Simulink communicate with external software, different technologies can be used as explained in the following chapter 4-Matlab Simulink for Co-Simulation. The simplest system is where a file is used where a model of a co-simulation system can write and read to like described in the paper [16]. The functions `fread` and `fwrite` of Matlab can be used to write and read to a file in order to exchange data be-

tween two Matlab instances. But a more frequently used approach is to use a TCP/IP connection to exchange data in a co-simulation system such as in the C-based simulations explained in [17]. For rapid testing and analysis the NASA's Trick Simulation Environment [19], which is a simulation development framework developed at NASA JPL, was extended to pass data between the Trick simulation and the flight software modeled in Simulink with a TCP/IP connection implemented in custom S-Functions.

In addition to software-in-the-loop simulations, Matlab Simulink can export the model into code for various reasons, such as rapid prototyping which is described in [18]. The paper explains an algorithm which was implemented in Simulink and exported to code to run on a microcontroller. Only small adjustments were made to the interface of the Simulink model in order to work with the microcontroller. Complex tasks, that would be troublesome to code by hand, can effectively be generated to code. Reports are generated too which show the direct connection between the code and the Matlab Simulink model blocks. This helps to understand the generated code more easily, and is a documentation of the code as well.

In the paper by González F., et al. [8] different coupling methods were researched in terms of their performance for connecting a C++ software with Matlab Simulink. Three methods for coupling an external software with Matlab Simulink were identified and tested. The three methods were firstly importing and exporting a data file in order to exchange data. The second method are function evaluations where functions are executed on the external software and which return results in order to extend the simulation capabilities of the software. The third method is where two simulation software systems share data at defined time steps that are synchronized between them. Tests were conducted to identify the speed of the different methods. It was identified that the first option with the data file was the one with the most overhead and slowest execution speed. The second method for function evaluation showed three technologies in order to communicate with, the Matlab Engine, Matlab Compiler and the MEX API (S-Functions), in which the Matlab Engine was the slowest, the Matlab Compiler was the second fastest and the MEX API was the fastest method. But each method for function evaluation has a different degree of development effort with MEX API requiring the highest effort for implementation. All three methods were researched in this thesis in chapter 4-Matlab Simulink for Co-Simulation. In the third option of the co-simulation approach a TCP/IP connection was used to connect the Matlab Simulink simulation with the C++ software as well as exporting the Matlab Simulink model to code into a Dynamic Linked Library (DLL). The approach to export the software into code was the fastest approach as the code can be executed directly by the C++ software, but is also more complex to implement than to use a TCP/IP connection to the model.

# 4 Matlab Simulink for Co-Simulation

In the future manufacturers want to use their own implementation of a shift strategy and additional components for the VECTO simulation. A model of such a component is being designed in the software Matlab Simulink. Simulink provides many extensions and out of the box implementations to communicate with external software or hardware. As a part of this thesis a research of different possible solutions to communicate with external software during a simulation of VECTO was conducted. The model needs to acquire certain vehicle parameter data before the simulation starts. During the simulation at every step, all the data that is necessary needs to be sent to the external component and a result is being returned back to VECTO.

## 4.1 Overview Communication Technologies Between Matlab Simulink and C#

In this chapter the most promising technologies to communicate with software outside of Matlab Simulink were researched. The following Table 4.1 gives an overview of the technologies to send and receive data between Matlab Simulink and an external application. The technologies will be described in the following sections.

### 4.1.1 Run with Matlab

These technologies can be divided into two different categories. The technologies in the first category is able to be executed while Matlab Simulink is running whereas the second category are technologies which are being able to run on their own outside of Matlab Simulink to send and receive data to and from an external software. Each category has their benefits and drawbacks. The most important difference is when being run with Matlab Simulink all the features that Matlab Simulink provides can be used. In contrast, when running separate from Matlab Simulink some features or functions may not be available.

**Table 4.1:** Matlab and C# Connections

| Type | Name | Info | Connection | Multithreading |
|---|---|---|---|---|
| Run Separate from Matlab | Simulink Coder/Embedded Coder | Generate C/C++ Code | Wrapper for C++ Code to access with C# | Yes |
| | | Shared Library | Include dll in C# Application | Yes |
| | Simulink Compiler | Generate Standalone App/FMU | Interface with FMI, Wrapper for using in C# | Yes |
| Run With Matlab | COM Interface | Execute Matlab function from C# | Matlab function starts Simulink mode, C# starts Matlab function | Maybe[1] |
| | Engine Applications | Matlab Engine C++ API | Control Matlab engine over C++ API | Maybe[1] |
| | Access .NET Library | Import C# DLL in Matlab | Define communication in C# DLL. Inter-Process Communication, IPC, files, pipes, Sockets, RPC, Shared Memory | Maybe[1] |
| | HTTP Interface | HTTP Communication | Use Matlab RESTful web services to access HTTP GET and POST methods | Maybe[1] |

[1] Matlab is single-threaded but multiple Simulink Models can be started in parallel and on separate threads with workers with the Parallel Computing toolbox (without the toolbox they run in serial)

### COM Interface

The Component Object Model (COM) is an old technology developed by Microsoft for interprocess communication. A COM object has an interface with properties, methods and events. It is exposed by a COM server which can then be accessed by a COM client. Matlab can either be a COM server or client. That is why an application and Matlab Simulink can also be client or server to communicate with each other. [27]

In practice this means that an application which acts as the client can call Matlab as the server and execute a function in Matlab. Data can be passed as parameters to the function and a result is being returned to the client. Additionally, variables, matrices and character arrays can be returned from the Matlab workspace. A workspace in Matlab holds all the currently known variables of different data formats. Furthermore, Matrices and character arrays can be sent to Matlab and stored in the Matlab Workspace. [27]

The interface that is provided by the Matlab Simulink COM Server does not have many features. It is possible to execute functions and pass some data to the client. In addition, it is not possible to create a custom interface and use it in Matlab Simulink as a COM Automation Server. This means the available functions can only be used, but not extended. [28]

Vice versa, when Matlab operates as the COM client and an external application is the COM server, it has more advantages and more features are available to communicate with each other. COM objects can be registered in Matlab as an executable or a Dynamic Link Library (DLL). For example, Microsoft Excel offers a COM interface which can be registered in Matlab with the Excel executable. Functions provided by the COM server can be executed in Matlab. Additionally, get and set functions retrieve data from the COM server or send data to the COM server. Events can also be raised by the COM server which means a client registers to an event and can then provide an action on what to do if it is triggered. In Matlab a function is attached to an event, which will be executed when the event is triggered. [29]

### Engine Applications

An engine application is a program that uses the Matlab Application Programming Interface (API) to call functions in Matlab from an external software. C, C++, Java, Fortran and Python APIs are supported. Similar to the COM interface, functions in Matlab can be called with parameters and results being returned. Additionally, more data can be exchanged between Matlab and C++ with the MATLAB Data API. [30]

Matlab can either be started from C++ synchronously, asynchronously or a C++ application can be attached to an existing shared Matlab session also either synchronously or asynchronously to start evaluating commands, run functions or pass data to Matlab.

Asynchronously means multiple processes are interacting with each other and therefore can do work independently. The C++ API can also be used by different threads, and multiple shared Matlab sessions can be accessed or multiple threads can connect to a single Matlab session. [31] [32]

Many different data types can be transferred from C++ to Matlab via the Matlab Data API such as various array types (Cell array, character array, et cetera), enumerations, strings, structs and more. [33]

The Matlab Engine API for C is similar to the C++ API. It communicates with the Matlab Engine through pipes on Unix platforms and the COM interface on Windows to send and receive data, execute commands and start a Matlab process. The C Matrix API only works for Matlab 2017b or earlier versions and it is recommended to use the C++ API instead. [34] [35]

### Access .NET Library

The .NET Framework class libraries can be used in Matlab, either the libraries that are already installed on the system or by loading a .NET assembly. A .NET assembly can be loaded by their global name when it is registered as a global assembly on the system or with the full path to the file to load it as a private assembly. [36]

After loading the assembly, the methods and properties of the classes in the assembly can be used or accessed. Thus, the methods, properties and delegates of .NET can be used, which tell C# to call a method when an event is triggered. However, there are some limitations in using a .NET assembly such as that arrays can be passed to methods of an assembly, but structure arrays, sparse arrays or complex numbers cannot be passed as parameters. All limitations can be looked up in the documentation of Matlab at [37, Matlab Documentation].

### TCP/IP and HTTP

There are a few options to use HTTP requests and responses in Matlab Simulink. On the one hand it is possible to use Matlab code to send HTTP requests and to receive HTTP responses to execute Matlab code in Simulink. On the other hand there is the option to use the TCP/IP Send and TCP/IP Receive Blocks of Simulink in order to send and receive data during a Matlab Simulink simulation. This can be done during the simulation or at certain time steps and it can either halt the simulation or send and receive the data asynchronously. [38] [39]

Another option is to use C/C++ Code and establish an HTTP connection to VECTO to send and receive data during the simulation. This can be done in various ways. The simplest method is to include C/C++ code in the Matlab Simulink model and

execute the code with the function `code.eval`. With `code.eval` the C/C++ function is executed and data can be passed with the parameters and returning a result value. In C/C++ code a HTTP connection can then be established to VECTO, passing on the parameters and waiting for the result.

Another way to use C/C++ code in Simulink is making use of an S-function block. An S-function block is a custom Simulink block that can be programmed in different programming languages. S-function blocks are described in more detail in the chapter. 4.1.4 S-Function Block.

### 4.1.2 Run Separate from Matlab

The technologies described in the following sections are different ways to execute the model outside of Matlab Simulink. One of the options is to only install a Matlab runtime, which enables standalone applications or components to be run, without the whole Matlab and Simulink installation and just with the runtime. However, this does not enable all features of Matlab. Additionally, for some applications the correct runtime version or correct Matlab version is necessary to be installed. With every version new features are added or some features are deprecated and cannot be used anymore. The following technologies do not have these drawbacks as they work without any installation of Matlab and Simulink or the Matlab runtime. [40]

#### Generate C/C++ Code

The option to generate C/C++ code from a model exports the whole model, including all blocks, functions and S-Function blocks as long as they are supported. The ability to export the model into C/C++ comes with a few drawbacks, as many features that are available in Matlab or Simulink cannot be used. There is a list with all the functions and objects that can be used in code generation Simulink models. In addition, some language features of Matlab are not supported which are listed in the following links at [41, Matlab Documentation - Functions] and [42, Matlab Documentation - Language Features Supported].

The model can be configured for certain targets to be compiled for and the language in which the model should be compiled to as well as some code generation objectives. These system target files declare which target the code generation should be compiled to. These are called *grt.tlc*, *ert.tlc*, *ert_shrlib.tlc* or any other custom target file. A system target file produces code in accordance to the target hardware and operating system. Thus, each type of system target file has certain code generation features available. Some system target files require certain Simulink toolboxes to be installed in order to use them. There are many more configuration possibilities for code generation in the settings such as specifying which math library to be used or generating a C-API

for signals, parameters, states or root-level I/O and settings for support of floating-point numbers or non-finite numbers. The resulting code can only be written in C or C++. The code generation objectives can be chosen whether the result model should be compiled for debugging or for efficiency. This is helpful when searching for an error in the resulting code or during the execution of the model. [43] [44]

The *ert.tlc* target file uses the Embedded Coder Toolbox and the *grt.tlc* target file uses the Simulink Coder Toolbox. The Simulink Coder generates code from Simulink models and the Embedded Coder extends the Simulink Coder and Matlab Coder. The Matlab Coder generates code from Matlab code. The Embedded Coder has optimizations and more control over the resulting code for embedded processors. It also offers support for the AUTOSAR, MISRA C and ASAP2 software standards and helps the code generation with reports, documentation and verification of the code. Additional packages for the Embedded Coder are available to support specific hardware. [43] [45] [46]

The resulting code includes project files for Microsoft Visual Studio, to automatically open the code in Visual Studio and to work with it or including it in other projects. In Addition to the resulting code and Visual Studio project file, a custom make file is included to compile the source code into a library or executable that is explained further in the following section.

### Shared Library

After exporting the Simulink model into C/C++ code, the resulting code can be compiled into a Shared Library, which is a DLL that can be loaded into other projects or in Matlab itself. Besides a DLL for Windows, for UNIX a shared object (*.so*) is generated and for MacOS a dynamic library (*.dylib*). These libraries are generated either for the whole model or a single component of the model. They can be included into existing applications and help to reuse code, add additional features to a software or hide intellectual property, where otherwise the source code of the component would be visible for anyone to be inspected. DLLs can still be reverse-compiled with special tools, to inspect the source code behind these libraries, but it is harder to read the source code. There are options to compile libraries with additional protection with randomization and encryption to make it more difficult to retrieve readable source code by reverse-engineering the libraries. These settings are used, when intellectual property of a model needs to be protected. [47] [48]

### FMU/FMI

Models that are exported to C code and shared with others or being used in other simulation environments, have the problem to define all the inputs and outputs of the

model and how to interact with the model to run it. The Functional Mockup Interface (FMI) describes an interface standard for simulation tools. It provides functions to communicate with models, setting the inputs, reading the outputs and reading or setting the parameters in the model. With this standard multiple models can be simulated and an algorithm can exchange the data between these models. The FMI standard provides 2 options, Co-Simulation and Model Exchange, which means that a model can be imported in other modeling environments and work further with it. The following informations about FMI are from the FMI Documentation for version 2.0 [49].

The Functional Mockup Unit (FMU) is in a simplistic way a wrapper over a Shared Library, which is a single model in the FMI standard. It is still a Shared Library, but with additional code of the FMI standard for inputs, outputs, parameters and functions to run the model. An FMU can be used in different target environments, such as Windows, Linux or Macintosh OS, as no specific header files are used. This allows to exchange the FMU cross-platform.

An FMU consists of a single ZIP file which contains the following data:

- XML file
- Binary files (DLLs)
- Source code
- Additional files

The XML file contains information about the model, its inputs and outputs, parameters and FMI specific settings. An FMU can be a Model Exchange or a Co-Simulation based FMU, or both. Additionally, the Unit Definitions and Type Definitions described in the XML file are further used for the model variables section. The model variables section describes all the variables that are accessible to be read or set in the FMU. The Model Structure of the model contains the inputs and outputs of the model, the continuous-time states and the initial unknowns. The model variables are the internal variables of the model and do not have to be visible to be set and read from. Only the inputs and outputs of the model are visible and they are the only way to communicate with the model. The initial unknowns are variables where the initial value is not known during the initialization phase of the simulation. The Default Experiment section describes the start time, stop time, tolerance and step size for the model. These values can be used by the tool that imports the FMU or can be ignored and custom values being used instead.

The Model Description section of the model contains the used FMI version. For this thesis the version 2.0 of the FMI standard is used, as it is the official stable version. The version 3.0 is still in beta, but being actively worked on and should be used in the future when it is ready, as many useful features are being implemented in this version.

The section contains information about the model name, author, model version, license, the used generational tool and more. The Global Unique Identifier (GUID) is a unique hash to identify and check whether the XML file and C code are compatible with each other. An input port or output port can have a Unit Definition which tells which unit it is, and it is being checked when ports are connected. This happens when the connection equations hold and the units are compatible. If this is the case, the model works but if an equation does not hold, an error may be shown by the tool that executes the FMU. The Type Definitions describe new custom types and they can either be a Real, Integer, Boolean, String or Enumeration. The model variables can have the same unit types as described before in the Model Description section and list all the variables in the model, that can be accessed. A model variable can have an initial value, the variability during the simulation and the causality which is a parameter, an input or an output. The variable can be set with the functions provided by FMI. There are certain rules which variables can be set in the instantiated, initialization phase, during simulation, at an event or at every communication point or whether they are constant or not.

A Co-Simulation FMU goes through certain phases during the simulation. The first state is called instantiated. At this phase the initial start values of the variables can be set, but only variables with the initial "exact" and "approx." setting. In the next phase, the initialization mode, the variables with the variability "input", which are the input variables of a model, can bet set. After exiting the initialization mode the simulation can be started. With the function `fmi2DoStep` a communication step is being performed. After the step is completed, the state of the FMU is either fmi2OK or an error is thrown such as step failed or step canceled. If the step succeeds the outputs and parameters of the model can be read, new inputs or parameters can be set and the next step can be performed. The simulation is finished, when either the maximum number of step sizes is reached or when the tool or master algorithm decides to stop the simulation. The FMU can be terminated and the final solution can be read. Some function calls are not allowed in every phase of the calculation similar to reading and setting variables. A list with the functions and when they are allowed to be executed can be found at [49, FMI for Model Exchange and Co-Simulation version 2.0 documentation - 4.2.4].

A Model Exchange FMU is here "to numerically solve a system of differential, algebraic and discrete-time equations" [49] as described in the documentation. The Model Exchange FMU has different phases than the Co-Simulation FMU. There is the initialization mode, the continuous-time mode and the event mode. Similar to the Co-Simulation FMU in the Initialization mode the initial values for the continuous-time states and for the previous discrete states can be set. In the continuous-time mode the values of the continuous-time variables are computed with the ordinary differential and algebraic equations. The discrete-time variables are fixed in this state and the discrete-time equations are not evaluated. In the event mode, new values for the

continuous-time variables and the discrete-time variables, which are activated at an event, are computed. In the end, the FMU is terminated and the solution at the final time can be read.

### 4.1.3 Matlab Code in Simulink

Using Matlab code in Simulink to utilize the COM interface or .NET Framework assemblies is not trivial and very limited to use. A Matlab script Simulink block can be used to execute Matlab code in Simulink, but it has only a subset of the Matlab script language functions available to use. The code of the Simulink block is exported in C/C++ code, but not all functions of Matlab are translatable into C/C++ Code. However, in every release of Matlab Simulink more functions are supported for code generation.

Matlab functions in a Matlab code Simulink block can be set extrinsic with the function `coder.extrinsic`. This means these extrinsic function are only executed during simulation, but not when they are exported with code generation. When Simulink model is exported to a standalone application, all the code that is set to be extrinsic is omitted from the resulting C/C++ code.

Another option is the interpreted Matlab Function block to execute a Matlab script in Simulink and to receive the results. This is different from the Matlab script Simulink block that was described before as the Matlab interpreter is used. A Matlab script .M file needs to be created with the Matlab code, which can then be executed by the Simulink block. But this is rather slow, as the Matlab parser is called during each time step to parse the file. This option does also not work when compiling the Simulink Model to code, it only works when Matlab Simulink is running and the simulation is started in Simulink. [50] [51]

### 4.1.4 S-Function Blocks

S-Functions are custom blocks in Simulink that can be programmed in Matlab, C, C++ or Fortran to extend the functionality of Simulink. These S-functions are compiled into MEX (Matlab executable) files and utilize the S-function API to communicate with the Simulink engine. An S-Function has inputs, states, outputs and parameters. There are also different types of S-functions which differentiate from each other on how the S-function should be implemented and on the set of features that are available for the specific type. [52] [53]

The following types of S-functions exists:

- Level-2 Matlab S-function

- C-MEX S-Function
- S-Function Builder
- Legacy Code Tool

The Level-2 Matlab S-function and the C-MEX S-Function types need to be written in code and both have certain features. They also have limitations, that the other type does not have. Like their name implies, the Level-2 Matlab S-functions are written in the Matlab scripting language. A large set of the S-Function API can be used but not the full one, the C-MEX S-Function can utilize the full S-Function API. The functions available for the Level-2 Matlab S -Function can be looked up at [54, Matlab Documentation]. The model simulation in Matlab Simulink works with S-function blocks but when code of the model is generated, a Target Language Compile (TLC) file is needed. With a TLC file, generated code of the block can be customized. There also exists the Level-1 Matlab S-function option, but it is deprecated and has a very reduced feature set. It is advised to use the Level-2 Matlab S-function instead as it can use the S-function API with much more features. [55]

The C-MEX S-Function is a custom Simulink block that is written in C. As with the Level-2 Matlab S-Function, some callback methods need to be implemented that are used by the Simulink engine during the simulation. For example, there is the method `mdlOutputs` which computes the output of the block. It does not matter in which function of the block the output is calculated, but the callback method is needed by the Simulink engine. The number of callback methods that can be used is larger than that of the Level-2 Matlab S-function. Some callback methods are required to be implemented for the C-Mex S-Function to work properly during a simulation. The required callback methods are `mdlInitializeSizes`, which defines the number of inputs and outputs and their dimensions of the Simulink block. The method `mdlInitializeSampleTimes` defines the sample time and offset time when the block is executed. The method `mdlOutputs` is called by the simulink engine each time to calculate the output of the function, and the method `mdlTerminate` is called when the simulation has ended and some tasks are still needed to be done. When this C-MEX S-Function is compiled with the MEX utility, a MEX file is created that can further be used to import the block in Matlab Simulink. [56]

C-MEX S-Functions are written in C, but they can also be written in C++. The major difference is that with C++ the callback methods need to be defined with C call conventions. The Simulink engine only works when the callback methods are in C calling conventions format. This can be done with the `extern "C"` statement. [57]

The S-Function builder and the Legacy Code Tool are similar to the C-MEX S-function, as they are also written in C, but help the user to easier use custom Simulink blocks and the features that S-functions provide. Both tools are even more limited in avaiable features, than writing the S-function in C code and compiling it to a MEX file. The

S-Function builder is a graphical user interface where C/C++ code can be set with additional specifications to create a C-MEX S-Function. In the S-function builder different types of states such as discrete or continuous state, the number of input and outputs and their data type and dimensions can bet set.

The Legacy Code Tool is the easiest way to use S-Functions in Simulink to have custom Simulink blocks, but has the most limitations in features. With specifying the existing C/C++ code and some options, the code can be converted into an S-Function. A TLC file can be created for code generation as well in the Legacy Code Tool and in the S-Function builder. [58] [59]

### 4.1.5 Other Tools

In this section, additional tools are presented which do not look viable for a connection between VECTO and a Matlab model, but are still worth mentioning.

#### Building Controls Virtual Test Bed (BCVTB)

BCVTB is a co-simulation software environment to couple different software or hardware together. Some programs supported by BCVTB are FMUs, Matlab and Simulink and more that are listed on the website. BCVTB is based on Ptolemy II which is a graphical modeling environment such as Simulink to define models. In Simulink the BCVTB block can be added, which connects to BCVTB during the simulation. Additionally, FMUs can be imported in BCVTB to be simulated and connected with other tools or FMUs.

BCVTB is not viable for VECTO, as VECTO is not the master application to control the simulation, but would have only a connection to BCVTB. [60]

#### WSDL-XML Interface

Web Services Description Language (WSDL) describes a web service and its server operations, arguments and transactions. It sends XML format messages with the Simple Object Access Protocol (SOAP) over Hyper Text Transfer Protocol (HTTP). Matlab is a client and a server provides functions and services The servers APIs can be accessed and data being sent over the function parameters and receiving results. [61]

## 4.2 Matlab Simulink Data Import

In order to load additional data into the model during simulation only a few options are possible, especially for dynamically loading data during the simulation. This means

data is being loaded, where the size of the data is not known in the beginning. As the available functions in Matlab Simulink are limited to use when generating code out of a model, so are the possibilities to dynamically load data. There are a few Simulink blocks which generate data such as the Sine Wave block. Data that is not yet known but needed during the simulation is necessary for a connection between VECTO and Simulink. [62]

### Root-Level Ports

Root-Level ports are the main technique to load data into the model. There are input and output ports. These can also be called inports and outports. A model can have multiple ports and they can connect models with each other. The data that is imported to these ports and then used by a model can come from the Workspace, such as the base Matlab or model workspace. A workspace contains the currently loaded variables in the specific workspace. When loading or creating a variable in Matlab, it is added to the workspace of the base workspace. With the Root-Inport Mapper tool, data can be set for these inports. These can come from signals, which can be created manually with the Signal Editor. The data can also be imported in this tool from a spreadsheet, where the top row are the signal names, the first column is the time and the other columns are filled with data. The data can also be imported from a .MAT file or from a workspace and added to the specific inports. Once the data is set for the inports, it can be accessed during simulation in the model. A restriction for inports are that no bus data can be imported from external sources. Bus data are individual signals in a structure. But the individual signals can be imported separately and then merged together into a bus in the model itself. This is an option to import data to include data during simulation when it is being run in Matlab Simulink. With code generation these in- and outports are available to exchange data as well. [63]

### "From File"/"From WorkSpace" Block

Similar to the option in the Root-Inport Mapper tool to import data from a .MAT file, there is also the From-File block to import data from a .MAT file during the simulation into the model. A limitation is with version 7.0 of MAT files, which reads only array-formatted data and some variables are handled differently than the new version 7.3 MAT file version. Additionally, the 7.3 version loads data incrementally during the simulation. Array-form data also only reads in double signal data. Additional limitations are for models that are being generated to code, as the block only allows "nonempty, finite, real matrix with at least 2 rows" as specified in [64, Matlab Documentation]. The data needs to be of type double and is not allowed to have any NaN, Inf, -Inf elements in the matrix which are often used as a replacement for values that are not available. The method to load data with the From-File block is recommended

by Matlab when loading large amounts of data, as the data is loaded incrementally with the newest MAT file version. [64]

### .XML and Other Custom Format Files

XML is not one of the supported file types of Simulink to load data into a model. However, there are other possibilities to load custom formatted data that is not officially supported. 3rd party tools are available to load a file in the desired file format. For example XML files are not officially supported, but there is an XML parser on the Matlab File exchange community. The File Exchange Community is a site on the official Matlab website, where people can upload tools and software they want to share with others for Matlab. [65]

### .JSON Files

In VECTO the vehicle data is exported in XML and JSON files. An official version of Matlab already exists to load JSON formatted text into Matlab objects. In addition to the official version, many 3rd party implementations are available to load JSON data into Matlab. The official version to load and export JSON data can be used, but one of the 3rd party implementations is JSONLab. There are some differences in the implementation in how the data is loaded into Matlab objects. The drawback of the official JSON parser and even the 3rd party tool JSONLab is that they cannot be compiled into C/C++ code and therefore not be used for a model that needs to be code generated. The 3rd party tool needs to be written in C/C++ code in order that the model can be code generated. [66] [67]

### DLL/S-Function Block

Another possibility to load custom data is a DLL which has code in C/C++ or add the code to an S-Function Simulink block. It is possible to make use of existing libraries that are written in C in order to load data in the Simulink model. The functions of the C/C++ code can be called directly with the Matlab function `coder.ceval`, when the source code files are added to the Simulink model. [68]

# 5 Design

This chapter describes the decisions which technologies to use in order to connect VECTO to a Matlab Simulink model during the simulation and to exchange data. Data of the vehicle needs to be loaded in the beginning of the simulation that is in a JSON formatted text file. During the simulation at every time step, data is being sent to the Simulink model which then returns a decision from the Shift Strategy.

## 5.1 Matlab Simulink - VECTO Connection Technology Decision

All the technologies that were researched in the previous Research chapter can be used to achieve the goal to connect VECTO to a Matlab Simulink Model during simulation. Some have advantages over others in their complexity to use, in their implementation and their speed of execution. The main category to differentiate between all the technologies is whether Matlab Simulink has to run during the simulation or whether the model can run independently of Matlab Simulink. The main disadvantages of running with Matlab Simulink are the complexity and related problems that are introduced when running another software with VECTO simultaneously. VECTO needs to be the master in the simulation process. Additionally, there can be differences between Matlab Simulink versions. New features and functions are added in every version of Matlab Simulink and certain features or functions are set to be deprecated or are not even available to use anymore. This means the correct version needs to be used by everybody who wants to use VECTO with a Matlab Simulink model or VECTO needs to support multiple Matlab Simulink versions which increases development effort. But the advantage of running Matlab Simulink during simulation, is that all features and functions of Matlab Simulink are available to use. When running independently of Matlab Simulink, not all features can be used as the model has to be generated into code. This reduces the feature set and some functionality needs to be added by the developer which adds more development effort to export the model from Matlab Simulink.

The technologies such as the COM interface, the Matlab Engine C++ API or the import of a C# DLL to Matlab Simulink would all work to control Matlab Simulink, the model and the simulation. The vehicle data and the data that is sent at every

time step, still need to be exchanged between VECTO and Matlab over a separate technology. This could be handled by Sockets, an HTTP connection or by reading in a file with the data. Either one of the three solutions to control Matlab Simulink and the simulation can be used, only minor differences between them exist. In the end it depends on the developer, with which technology he has the most experience with. The easiest solution to send data between Matlab Simulink and VECTO would be an HTTP connection. The Matlab Simulink model waits for an HTTP request from VECTO. VECTO sends the HTTP request when a decision for a gear shift is needed. The data could be as big as needed, as multiple HTTP messages can be sent.

In summary, the main advantage that Matlab Simulink would run simultaneously with VECTO, is that all features that Matlab Simulink provides are useable. Either the COM interface, Matlab Engine C++ API or the import of a C# DLL to Matlab Simulink can be used, but there is no big difference between them. They are three well known technologies and many online resources about them exist online. It only depends on which technology the developer is most familiar with. The disadvantage of these three technologies when running Matlab Simulink with VECTO is the added complexity and possible problems with version differences and this sums up to additional developer effort.

The decision to use a technology where the model runs without Matlab Simulink during simulation was chosen to be more independent of Matlab Simulink. Some organizations only have a specific Matlab version available and cannot switch to another version in years due to licenses. The issue that some features are not available to use when running independent of Matlab Simulink can be neglected, as some 3rd party libraries can be used instead or the features can be added with S-function blocks or C/C++ code if necessary. For this thesis a 3rd party library in C was used in order to add the needed functionality for loading JSON files into the model. As well as some code of the `AMTShiftStrategy` of VECTO was not possible to be reimplemented in Matlab due to time constraints and had to be rewritten. When an OEM has a model that is implemented in Matlab Simulink and that will be used in a real controller as well, this problem that not every feature of Matlab Simulink can be used exists too. Therefore, it is not a real disadvantage as the code need to low level as well.

The easiest option is that the model is being exported to a shared library or source code, which can further be loaded into VECTO to be accessed. The loaded code or shared library can be started by VECTO instantly without speed issues and problems that come with other technologies. With the shared library or the source code, it is also possible to execute the model in parallel. VECTO starts a separate thread for every run and executes them in parallel to be faster. The shared library or source code can then be loaded in every thread.

Every Simulink model of a manufacturer would need to adhere to a defined standardized interface which signals are being transferred and which signals are wanted as a

result. The shared library or source code needs to have functions in order to initialize the model, run every time step, pass data, return results and in the end terminate the model. There are functions that are added by the Embedded Coder of Simulink to initialize the model and step through the model. But the FMI standard is more extensive and has more possibilities to be used in the future. The model is compiled into a shared library and the extensive standardized code of FMI is being added and can be executed by VECTO during the simulation. The advantages of the FMI standard are that there are many other tools that support the standard. The FMI standard has many features but if not needed, only the necessary functions can be used and the rest when needed in the future. As the FMI standard is being supported by many organizations, it is also future-proof to use it. The next version of FMI 3.0 is being developed actively and includes even more features.

### 5.1.1 FMI/FMU

The FMI standard is being maintained by the Modelica Association Projects which is a non-profit, non-governmental organization. An FMU can be created manually or exported by a tool and this FMU can then be imported or executed by a tool. In Matlab Simulink the possibility to import and export FMUs was introduced with the version R2017b. Not every manufacturer has the newest versions of Matlab Simulink, therefore it is necessary to have a solution that works with older versions of this software as well. There are many solutions from different companies to import and export FMUs in both paid and free versions of Matlab Simulink. A few of them are the "Modelon FMI Toolbox for Matlab/Simulink" [20] which looks very promising, but is a paid version. Another solution is the "Easy FMI Add-On for Matlab/Simulink" [21] which can export and import FMUs for co-simulation in Simulink but is also a paid version. The most promising, easy and free solution is the "FMIKit-Simulink" by CATIA-Systems [22] that is being actively maintained on Github, where the source code can be inspected and changed if necessary. The person from CATIA-Systems who maintains the tool is actively working on the new FMI version 3.0 of Modelica too. An issue that came up during the development with the "FMIKit-Simulink" for this thesis was fixed very fast as well. All of these reasons are in favor to use the "FMIKit-Simulink" which also supports older Simulink versions upwards from R2012b to R2020a.

In practice, it is not very important which tool is used to export the model from Simulink into an FMU as long as it is a valid FMU that supports all the signals that VECTO provides and needs. The FMU can be programmed manually in C code with tools such as the "FMU SDK" by Synopsys [23] or the "FMU SDK" by Qtronic [24].

The "FMIKit-Simulink" can be added to Matlab Simulink and set as a target in Simulink for the code generation of the model. The resulting FMU ZIP file can then

be unzipped and the files in it used for loading the binaries of the FMU and calling its functions or looking into the resulting code.

### 5.1.2 C# Wrapper

Loading a shared library binary in C# is very common and used all the time. Every library or project written in C# is a Dynamic Linked Library (DLL), a shared library that can be used in other projects to reuse the code or organize a project more efficiently. A DLL can be imported in Visual Studio, the standard software tool by Microsoft to write C# source code. There are two main methods to load a DLL into a C# project. Either by adding it during development to the project and then compiling it. The other method is to dynamically load the DLL after the project was already compiled. The first method would not work, as for every FMU of a manufacturer VECTO would be need to be recompiled for it, as well as after every change in the FMU it would be needed to be recompiled. Therefore, loading the DLL dynamically is the only option. This can be done with the `DLLImport` attribute. There is also a difference between a DLL written in C# and a DLL written in C. A C# DLL is called a managed library and can be imported in every C# project. Every function of it can be called and the data structure can be accessed. The functions of a C DLL cannot be called in the same way. The function headers of the DLL need to be redefined for C#. Therefore a Wrapper is needed to call the C DLL in the correct way. In order to work easily with the FMU as a DLL in C# an existing project is being used that is called "FmiWrapper" [25] with a MIT License that allows to use the software free of charge as long as the MIT License is included with it. The "FmiWrapper" has already all the interface calls to the C DLL and provides easy to use functions in C# to call them. The project can be included into the VECTO C# project and a custom Shift Strategy class is created that gets all the function calls instead of the original Shift Strategy. The custom Shift Strategy then calls the FMU with all the parameters during each time step.

### 5.1.3 Structure Overview

The overall structure of the connection between VECTO and Matlab Simulink models is depicted in Figure 5.1. VECTO calls the new custom Shift Strategy instead of the original one. The new Simulink Model Shift Strategy calls the FMI Wrapper with the correct parameters. The FMI Wrapper calls the DLL of the FMU, which is then executed and returns data back to VECTO over the FMI Wrapper and the Shift Strategy. This is the same concept that will be used for the Hybrid Strategy or can be applied for any other component that can be replaced with a Matlab Simulink model.

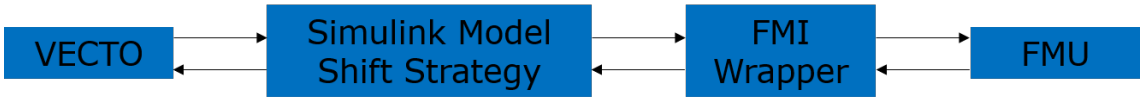


**Figure 5.1:** Structure of overview of VECTO and FMU connection

The structure is simple and not overly complicated. One important point is that all the signals and data that are passed between VECTO and the FMU needs to be defined and standardized, so the FMU can get all the data that are needed.

## 5.2 Data Interfaces

The decision on how to exchange the data that are needed is not very complex. There are only two types of data. The vehicle data that are available as a JSON formatted text file and the data that are exchanged during each time step to the FMU and the data that are returned to VECTO.

### 5.2.1 VECTO Signals Root-Signal Ports

The data that are being sent from VECTO to the FMU are the function parameters in the Shift Strategy of the function `ShiftRequired`, the function `InitGear` and the function `Engage`. In addition to these variables, the data that are in the `Databus` are also needed in the FMU. The `Databus` contains data of all the components such as the gearbox, axle gear, engine, vehicle, mileage counter, clutch, brakes, wheels, driver and driving cycle. All of these data will be sent to the Root-Signal ports of the Simulink model. The signal ports need to be created in the model in order to access the data. When the input ports exist in the model, they will be in the XML file in the FMU. The XML file is then being read in in VECTO and all the available inputs are known. The name of the inputs can then be matched with the data of VECTO and being sent to the FMU.

### 5.2.2 Loading Model Parameters for FMI Model

As the vehicle data is available in a JSON formatted text file it can be read into the Matlab Simulink model. This cannot be done over the Root-Signal ports of a model. The data have to be loaded separate from the signal ports. In order to load JSON files from VECTO easily into Matlab Simulink the library "cJSON" [26] was used, which is a lightweight JSON parser written in C. In order to have more control over the loading of the data and interaction between Matlab Simulink and the "cJSON" library a small wrapper was used in between these too. The wrapper functions call the

"cJSON" functions to load the data and the Matlab Simulink code calls the wrapper functions. Instead of using S-Functions, the function `coder.ceval` was used to call the C functions and pass parameters which data was needed and to return the data, in order to keep it as simple as possible. In order to save the data in between time steps of the model, the pointer address of the data is saved in the model and passed back during each run to the C wrapper to get additional data of the JSON structure.

# 6 Implementation

In this chapter the implementation of how VECTO will connect to a Matlab Simulink model is being described. The first part is about the generic AMT Shift Strategy in the official version of VECTO with the version number 3.3.5.1812. In the second part the new hybrid version of VECTO is being used with a OEM Hybrid Strategy that is implemented in Matlab Simulink.

## 6.1 VECTO Generic AMT Shift Strategy

There are 2 types of transmissions, manual and automated manual transmissions (MT/AMT) and automated transmissions (AT). The main difference is that the MT/AMT have a traction interruption when the transmission is not connected to the powertrain, that is simulated in VECTO during gear shifts. Whereas the AT does not have a traction interruption but still has some losses due to the clutch opening and closing. [7]

The AMT shift strategy was used to re-implement it in Matlab Simulink in order to test the FMI standard and overall concept. Furthermore, it was used to have a comparison to the original implementation in VECTO and whether some differences between them would occur, or whether they produce the same results. As can be seen in Figure 6.1, VECTO is calling during its simulation the Shift Strategy. The customized Shift Strategy will forward the request to the FMI Wrapper that is split into 3 components in order to forward the request from C# to the C code of the FMU. The FMU is the reimplemented AMT shift strategy of VECTO as a Matlab Simulink model.

### 6.1.1 Re-implementation of the AMT Shift Strategy in Matlab Simulink

The AMT shift strategy is called after the internal combustion engine returns a success. Some parts of the gearbox and engine components are re-implemented in Matlab Simulink as well due to that the gearbox is deciding which gear would be better to shift to. Therefore, the data for the operating points for the other gears have to be calculated and VECTO does dry-run requests over the gearbox to receive this data.

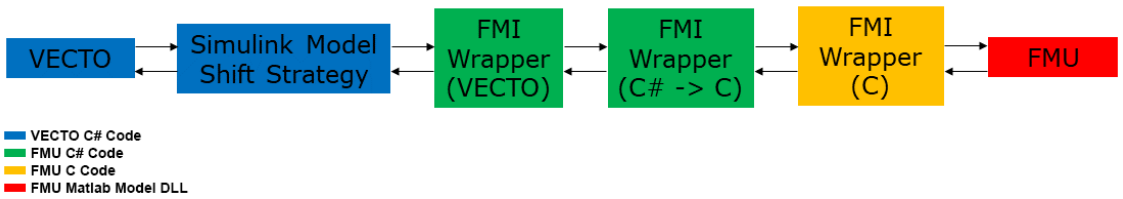


**Figure 6.1:** Structure of the VECTO and the FMU model

In the `AMTShiftStrategy` class, the constructor and the functions `NextGear`, `Engage`, `Disengage`, `InitGear` and `ShiftRequired` are the main functions and public and therefore can be called by other classes. In order to keep the logic simple and to avoid rewriting the functionality and logic, the function names and logic flow are staying the same as far as it is possible in Matlab Simulink. The `NextGear` function just returns the internally saved next gear, which doesn't need to be forwarded to the model as it can be saved after the function `ShiftRequired` returns a new gear shift. The function `Disengage` is an empty function as it is not used in the `AMTShiftStrategy`. The 3 functions `InitGear`, `Engage` and `ShiftRequired` are the 3 main functions that are needed to be reimplemented in the Simulink model.

### Data and Signals

The first important step was to decide on all the signals that are needed for the connection between the model and VECTO. The parameters of the 3 functions that need to be implemented are:

- absTime
- dt
- OutTorque
- OutAngularVelocity
- InTorque
- InAngularVelocity
- Gear
- LastShiftTime
- LastUpshift
- LastDownshift

The `absTime` is the current time in the VECTO simulation. `dt` is the length of time of the current simulation intervall. The `OutTorque` and `OutAngularVelocity` are the

torque and angular velocity that is going out from the current component. Similar to the `InTorque` and `InAngularVelocity` only are these the values that are coming in. The `Gear` is the current gear. The paremeters `LastShiftTime`, `LastUpshift` and `LastDownshift` are from the gearbox the last shift time, the last time it was upshifted and the last time it was downshifted.

```
public override uint Engage(Second absTime, Second dt,
   NewtonMeter outTorque, PerSecond outAngularVelocity)

public override uint InitGear(Second absTime, Second dt,
   NewtonMeter outTorque, PerSecond outAngularVelocity)

public override bool ShiftRequired(Second absTime, Second dt,
   NewtonMeter outTorque, PerSecond outAngularVelocity,
   NewtonMeter inTorque, PerSecond inAngularVelocity, uint gear,
    Second lastShiftTime)
```

These are the functions that are called in the AMT Shift Strategy from other components and the respective signals passed as parameters with it. The `Engage` function is called when the gearbox should is disengaged and should be engaged again, some values are set in this function for the AMT Shift Strategy. The `InitGear` function is called in the beginning of the simulation to initialize some values in the AMT Shift Strategy with the vehicle model parameters. The `ShiftRequired` is the function where the calculation happens whether a shift is needed or not. In addition to the function parameters the `Databus` is passed to the model and is needed to be specified as root-signal inports in Simulink. The signals that are present in the `Databus` are the following:

- GearboxInfo
  - Gear
  - TCLocked
  - StartSpeed
  - StartAcceleration
  - lastShiftTime
  - TractionInterruption
- AxleGearInfo
  - AxlegearLoss
- EngineInfo
  - EngineSpeed
  - EngineTorque
  - EngineIdleSpeed
  - EngineRatedSpeed
  - EngineN95hSpeed
  - EngineN80hSpeed
- VehicleInfo
  - VehicleSpeed
  - VehicleStopped
  - VehicleMass
  - VehicleLoading
  - TotalMass
  - CargoVolume
  - MaxVehicleSpeed
- MileageCounter
  - Distance
- Brakes
  - BrakePower
- WheelsInfo
  - ReducedMassWheels
- DriverInfo
  - DrivingBehavior
  - DrivingAction
  - DriverAcceleration
- DrivingCycleInfo
  - PTOActive
  - Altitude
  - RoadGradient
  - CycleStartDistance

The signals are based on the different components as implemented in VECTO: Gearbox, AxleGear, Engine, Vehicle, Breakes, Wheels, Driver and DrivingCycle. Each component has different signals that are provided in the Databus to lookup for other components if they need these values in the calculation. All of these will be sent to VECTO, but not all of them are needed for the Shift Strategy. However they may be necessary for manufacturers to know everything of the current state of each component. For the use for manufacturers these signals would need to be standardized and fixed.

The booleans such as `TCLocked`, `VehicleStopped` and `PTOActive` had to be passed as a number (double) and in the Simulink model cast back into a boolean, as the inports can

only be numbers. Similar with the enumerations `DrivingBehavior` and `DrivingAction` that are passed as numbers and in the model cast into the respective enums back.

In addition to the parameters and databus, the previous states of the gearbox and engine are necessary for the `AMTShiftStrategy`. These values are properties in the gearbox and engine classes and parts of these two classes need to be reimplemented in the Simulink model as well. However, the properties cannot be saved in the Simulink model and have to be passed as signals to the model as well.

- Gearbox
  - PreviousStateInTorque
  - PreviousStateInAngularVelocity
  - PreviousStateOutTorque
  - PreviousStateOutAngularVelocity
- Engine
  - PreviousStateEnginePower
  - PreviousStateEngineSpeed
  - PreviousStateEngineTorque
  - PreviousStateEngineTorqueOut
  - PreviousStateFullDragTorque
  - PreviousStateIntertiaTorqueLoss
  - PreviousStateOperationMode
  - PreviousStateStationaryFullLoadTorque
  - PreviousStatedt

The previous states need to be passed over to the model and cannot be saved in between time steps for the next one in the model. In every time step of VECTO the engine and gearbox components are called and calculations are happening, but the shift strategy is not called in every time step and therefore should not save the previous states. These previous states are needed for the AMT shift strategy calculation and were added therefore.

In addition to all these parameters an additional variable is passed over to the model which is a number to depict which function is being called in the shift strategy and to call the same function in the model. The inport signal for this is called `StrategyType`.

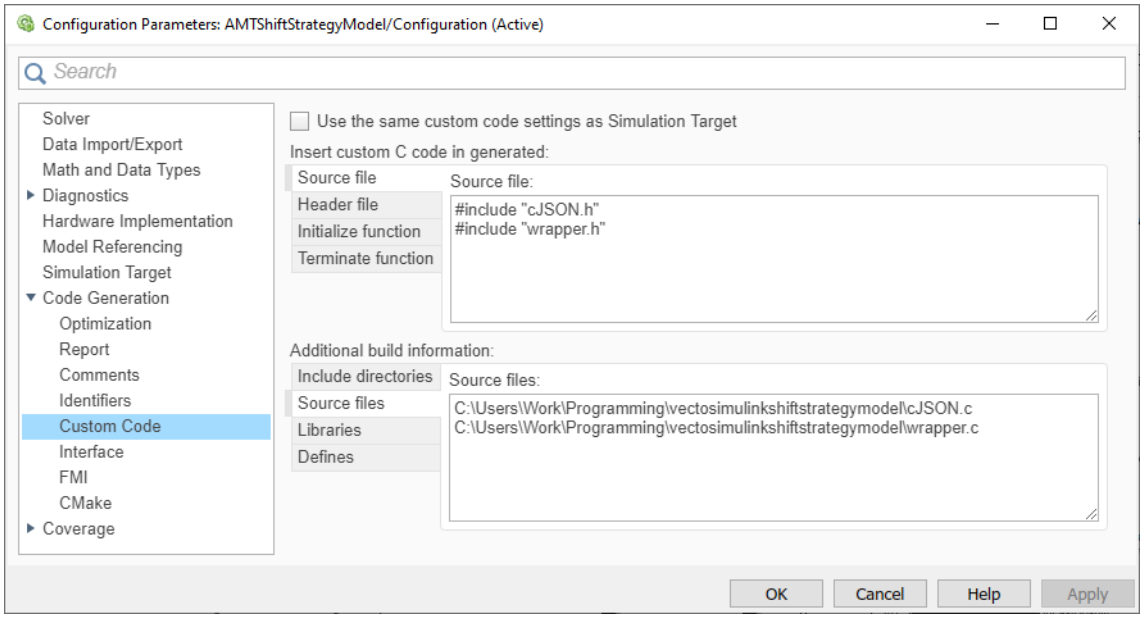


**Figure 6.2:** C source files added to the model in the Settings

The simulink models returns 2 ouputs. One is the `ShiftRequired` boolean that is returned by the function `ShiftRequired`, which is either true or false, depending on if a shift is necessary or not. The other output is the `NextGear`, that is a number and returned by the functions `InitGear` and `Engage` to which gear should be switched to next.

### JSON File Loading

These are all the inports and outports that are passed over during the simulation from VECTO to the Simulink model. The first thing that is happening in the model is the loading of the JSON formatted text file with the vehicle data. The file name is hardcoded in the model with the name `VectoJsonModelData.json` and needs to be at the same place as the executable that is running the FMU in order to be able to load it. This is due to simplicity as the in- and outports of Simulink do not yet allow using a string as a datatype for a port for the version Matlab R2020b and R2016b that are being used for this master thesis. In the newest version of Simulink this should be working, as it is described in the documentation of the inport.

The source code files of the small C wrapper functions, that are being used, are added in the settings of the Simulink model as can be seen in Figure 6.2. Additionally, in the Matlab code the files have to be added with the following functions to add the files to the build information for code generation. After this two settings all the functions of the wrapper can be used.

```
coder.updateBuildInfo('addSourceFiles', 'wrapper.c')
coder.updateBuildInfo('addSourceFiles', 'cJSON.c')|
```

The reading of the JSON file could be done in the C wrapper as well, but as some basic functions are still available in Matlab, it was done in Matlab code. The functions that are normally used in Matlab code to read text files cannot be used as they are not supported for code generation. The only functions that are supported for code generation are low level I/O functions such as `fopen`, `fseek` and `fread` to read in the JSON file as a character array. This character array will then be passed over to the C wrapper file, which calls the `cJSON` library to decode the character array into a JSON structure. This JSON structure is being returned as a pointer, as Matlab does not know how to read this structure. To interact with the JSON structure directly, the whole structure would need to be defined in Matlab. This would be very time consuming and overly complex as it is simpler to just use the C wrapper for passing the pointer back and using the functions of the `cJSON` library to access the JSON structure and receive the wanted variables.

```
double_value = double(0);
coder.ceval('getDouble', cJSON_ptr, coder.wref(double_value)
   , cstring(jsonName))
```

The function `coder.ceval` is used to call the C wrapper and pass the parameters. The `getDouble` is the function in the C wrapper that should be called. The `cJSON_ptr` is the JSON structure as a pointer that we pass to the C wrapper. The `double_value` is the result variable where the result is being saved to. Therefore the variable is passed with the function `coder.wref`, which passes the parameter as a writeable parameter. The `jsonName` variable is the JSON name of the value that is wanted. The function `cstring` adds a null character at the end of the JSON name, as it is standard for C strings. A JSON name that for example is being used is `AxleGearData.AxleGear.Ratio`.

There are two other functions for gathering data from the JSON structure, that are `getArray` to gather arrays in the JSON file like a lossmap of the gears, and the function `getItemList` to get for example the number of gears that are in `GearboxData.Gears`. In order to correctly get an array and the list, the size of these have to be known beforehand to save the array and list in a variable. Therefore, the functions `getArraySize` and `getItemListSize` return only the size of the respective array and list in the C wrapper. After knowing the size, the empty variable with the correct size for the array and for the list can be initialized and the data can be requested for them.

The JSON structure that was implemented in VECTO was not compatible 100% with the JSON format that is being used in the JSONLab software. These changes were done manually directly to the JSON files but should be implemented to the JSON implementation in VECTO in order to ensure compatibility and to directly pass the JSON file to the FMU. Numbers look like in the JSON file as `"SerializedValue": "0.0000 [1/s]"`,

but the unit needs to be separate and the number to be without quotes, in order that Matlab recognizes it as a number and not a string. Furthermore it can be used directly as a number instead of manually converting it from a string to a number. The same problem is existing with arrays such as the loss maps where the unit and numbers are in quotes:

```
"LossMapSerialized": [
    "0 [rpm], 10.0000 [Nm]",
    ...
```

This array needs to look like:

```
"LossMapSerialized": [
    [0, 10.0000 ],
    ...
```

Additionally, field values need to start with characters instead of numbers:

```
"GearboxData": {
    "Gears": {
        "1": {
        ...
```

Which needs to be changed to:

```
"GearboxData": {
    "Gears": {
        "G1": {
        ...
```

Another problem are arrays that do not have a character name to address them:

```
"AxleData": [
    {
        ...
    },
    {
        ...
    }
```

Which needs to be changed to:

```
"AxleData": [
    "Wheels1": {
        ...
    },
    "Wheels2": {
        ...
```

```
}
```

This has to be changed for `AxleData.Wheels` and for `EngineData.Fuels`. One last thing that has to be changed are empty arrays [] or [""] that need to be changed to null.

A major change that was introduced with the Matlab version R2017a was the "Dynamic memory allocation for unbounded arrays and large arrays" as defined in the release notes [69]. Without this feature an array has to have a fixed size, which has to be defined beforehand. Therefore, an implementation for Matlab versions before R2017a need to have a fixed size for all the arrays that are used.

The memory that is acquired in the C wrapper for the JSON structure that is passed as a pointer to the memory location, needs to be saved in between the time steps. Otherwise, the JSON structure is deleted after every time step and needs to be loaded again in the next, which adds up to a decent amount of unnecessary execution time. Instead, the pointer is saved in a Data Memory Store in Simulink in between every time step. After the last time step the pointer is passed again to the C wrapper to free the memory taken by the JSON structure.

For Matlab Versions before R2017a it means initializing a buffer that is big enough to hold the loaded data. The JSON formatted file with the vehicle data is about 500KB big. Such a file contains about 400.000 `int32` integer values and the current buffer size is set to 1.000.000, which can be set up to the max size of an `int32`, which is 2.147.483.647. But it could be bigger by splitting the data into multiple arrays. The function `getJSONList` and `getJSONArray` have a buffer size of 2048 at the moment and can be set higher too in order to load data of the JSON structure, such as the lossmaps of the gears. These measures for having fixed sized arrays, only let the model work with Matlab versions before the introduction of the dynamic memory allocation for arrays. It is advised to use Matlab versions R2017a and newer in order to have no problems with memory allocation and array sizes in the Matlab model.

### Implementation of the AMT Shift Strategy in Matlab Code

In order to replicate the AMT shift strategy in the Simulink model all the C# code was replicated in Matlab code. Some functions or language features that are available in C# had to be replaced with a code that does the same as the C# code, but in Matlab code. For example, in Matlab array indices start with 1 instead of 0 as in C# and in other programming languages. The ternary operator is sometimes used in VECTO and is not available in Matlab and therefore was replaced with simple If/Else statements. The Physical Quantity Pattern [70] is implemented in VECTO which is a design pattern for the SI unit system to calculate values in different units correctly. This helps and simplifies the development of physical calculations in VECTO. Implementing this design pattern in Matlab Simulink is beyond the scope of this work, therefore the

calculations in the Simulink model were done by converting some values into other units directly. Two helper functions were added to simplify the conversion between units such as `RPMtoRadPerSec` and `RadPerSecToRPM`, both units needed to be converted very often in the shift strategy and the related calculations.

Another part of VECTO that needed to be replaced was the torque loss calculation in the gearbox class. This function "computes the torque loss (input side) given the output gearbox speed and the output-torque" as documented in the VECTO C# code of the `GetTorqueLoss` function. It triangulates the points in a delaunay map to calculate the torque loss of the gear's loss map. The calculation uses C# language features such as LINQ query operations to sort the triangles with 3 points in a list. This algorithm would be more complicated to reimplement in Matlab code, therefore it was substituted with a bilinear interpolation. The differences in the results are minimal and do not affect the calculation and further program flow.

Some special cases exist that needed to be handled such as when the function `Engage` returns the next gear. The next gear is saved as a property in the class `AMTShiftStrategy` which is returned for the public function `NextGear`. This variable needs to be saved in the model, as the new shift strategy should only pass all data to the model and the model needs to handle the rest. Therefore, the Simulink Memory Store for `nextGear` is added to the model that saves the variable inbetween time steps.

In Figure 6.3 the model is shown how it looks in Matlab Simulink can be seen. All the inports on the left side with the `PreviousState` bus and `DataBus` bus, which consist of the all the individual inports, are joined together with the Bus Creator block. The big block in the middle is the Matlab Script block which contains the whole Matlab code of the AMT shift strategy. On the top is the `nextGear` and `cJSON` Data Memory Stores and on the right are the outputs.

### 6.1.2 Export of FMU in Matlab Simulink

With the "FMIKit-Simulink" that is imported into Matlab, the Simulink model can be exported to an FMU. On Github either the release can be downloaded and directly imported in Matlab or the source code of the "FMIKit-Simulink" can be downloaded and compiled. The latest version at the time of this writing is the version v2.9. But after this version some bugs have already been fixed on the `develop` branch of the Github repository. Therefore, it is recommended to compile the source code of the `develop` branch and use this version to export the model to an FMU. This version includes the FMI 3.0 beta as well, which can be used in the future. A guide to compile the source code can be found in the repository at `docs/build.md`. When compiled the "FMIKit-Simulink" can be added in Matlab with the commands:

```
addpath("FMIKit-Simulink")
```

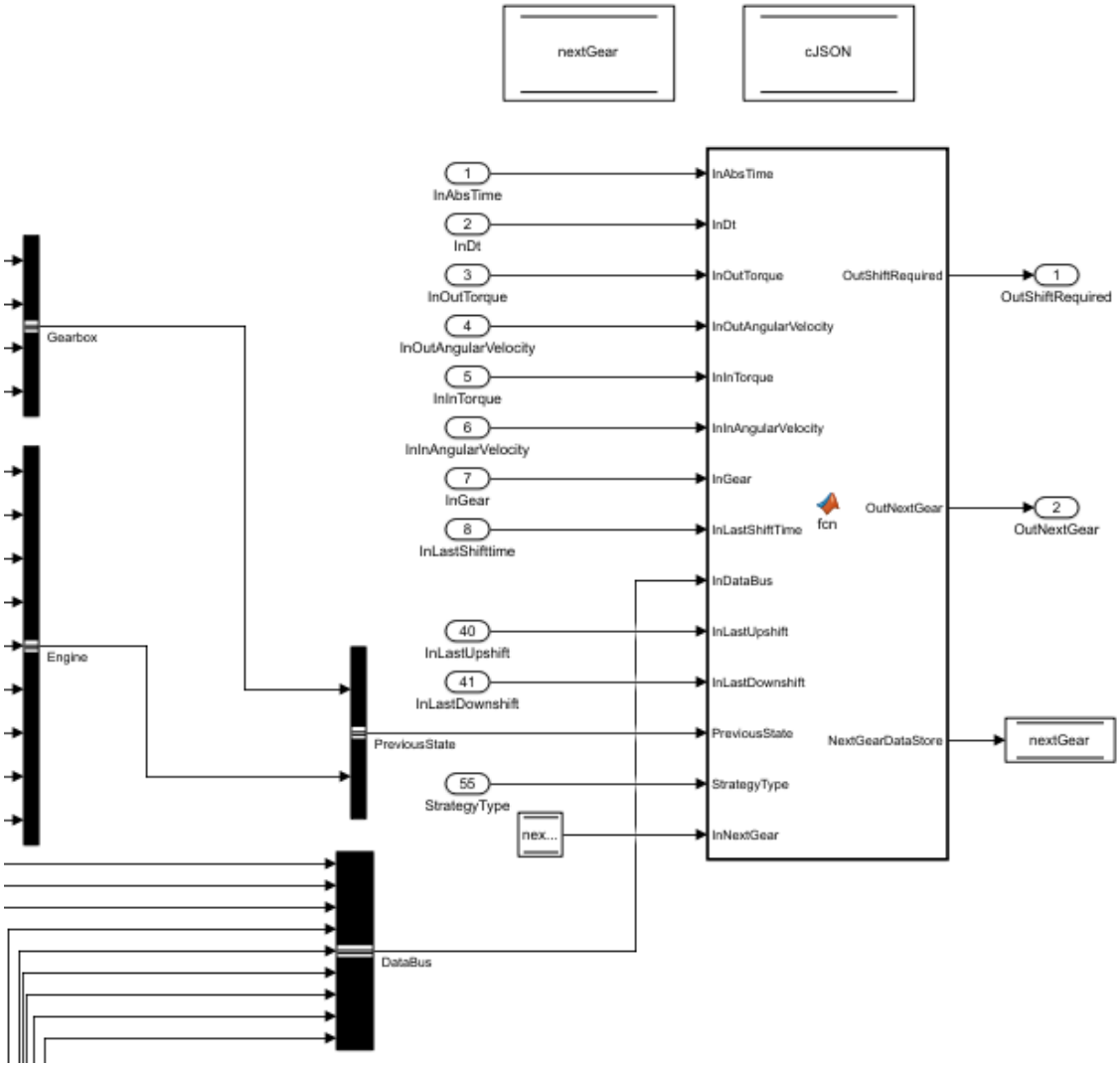


**Figure 6.3:** Graphical representation of the Shift Strategy model in Matlab Simulink

```
FMIKit.initialize()
```

The `addpath` command adds the folder `"FMIKit-Simulink"` to Matlab, which can be an absolute path such as `"C:\Users\Work\Programming\FMIKit-Simulink"` or a relative path. After adding the path the "FMIKit-Simulink" can be initialized and all the options for it are added to Matlab Simulink.

With the "FMIKit-Simulink" added to Matlab Simulink, the system target file can be set to `grtfmi.tlc` with the description `FMU based on GRT (Function Mockup-Unit)`. This setting can be found in the `Model Settings -> Code Generation -> Target Selection`. Additional settings can be set, such as "Select objective" which can be Debugging, Execution efficiency or Unspecified. In the settings `Code Generation -> FMI` the FMI version, visible parameters, model author and template directory can be set. The source code of the model should be included in the FMU. It can be very important to inspect the source code of the model, as the resulting C code includes comments of the Matlab code. Therefore, it can be inspected precisely how the Matlab code translates to the C code and whether any code is translated wrong, whether an error could occur or something is calculated wrong. In the following code listing an example Matlab code can be seen. It shows how it is translated to C code as well as how the function `RPMToRadPerSec` is included.

```
/* '<S1>:1:1365' deltaFull = totalTorqueDemand -
    dynamicFullLoadTorque; */
/* '<S1>:1:1369' response.DeltaFullLoad = deltaFull *
    RPMToRadPerSec(avgEngineSpeed); */
/* '<S1>:1:862' s = value * (2*pi) / 60; */

s->DeltaFullLoad = (i_double_value - dynamicFullLoadPower /
(double_value * 6.2831853071795862 / 60.0)) * (double_value *
6.2831853071795862 / 60.0);
```

It is different to build the model and generate the FMU for Matlab versions R2016b and R2020a. Building the FMU in the version R2016b can be started in the menu `Code-C/C++ Code-Build Model`. For the version R2020a the app Simulink has to be opened in the "APPS" tab of Simulink. A new tab opens with the name "C Code" where the model can be built with the button "Build" in this menu. In the build process a few checks are performed, one of them is a model check which can be set with the option "Check model before code generation" in the "Code Generation" settings. A report will be shown with possible warnings or error messages after the build is complete.

The resulting FMU can then be extracted like a ZIP archive and the files inspected. The resulting XML file `modelDescription.xml` contains the outputs and the inputs which can be checked whether everything was set correctly.

### 6.1.3 VECTO FMI Adapter

In order to import the resulting FMU in VECTO the DLL file needs to be loaded. This can be done with the C# attribute for functions `DllImport`, that is documented in more detail in the Microsoft .NET Documentation [71].

In order to reduce complexity a pre-existing project that already implemented the interface of the FMI standard in C and C# was used. This project on Github can be found at [25]. The project consists of a C project and a C# project. The C project handles all the memory allocation and freeing of loading the FMU DLL and provides the functions needed for the FMU in the C# project. The C# project provides then all the functions, types and definitions to be used in C#. This makes the task to import the FMU C# much easier, as all the complex calling is handled by the project.

In the Adapter the first thing that is done is to get the path to the FMU and to extract the archive. The XML model description file will be read in and parsed, as all the information in it is necessary to be known. The model description GUID needs to be known to be able to instantiate the FMU. As well as all the input and output names and the corresponding value reference need to be known in order to send data to the inputs and read data from the output ports. An input variable in the model description file looks like the following:

```
<ScalarVariable name="InAbsTime" valueReference="4" causality="
   input">
    <Real start="0" unit="s"/>
</ScalarVariable>
```

For this input port the value reference is the number "4" and with this the input port can be set each time step with the following C# code:

```
Instance.SetReal(4, new double[] { InAbsTime });
```

The variable `InAbsTime` has the data of the current time step of the AbsTime.

Yet before the input port data can be set, the FMU has to be instantiated with the following function:

```
void Instantiate(string instanceName, Fmi2Type fmuType,
   string guid, string resourceLocation, bool visible, bool
   loggingOn)
```

The `instanceName` is the model identifier and can be found in the model description file, as well as the FMU type that can be read in the file as well. The `GUID` has to be set as well and is unique for the FMU and changes when anything is changed in the FMU. With the `GUID` it can be checked whether the inputs and outputs stayed the same or have been changed. `GUID` shows as well if the FMU and the XML are

compatible with each other. The `resourceLocation` is a folder where additional data such as maps that need to be loaded can be found. The parameter `visible` determines to reduce the interaction with the user to a minimum or to show plotting windows or other application windows. The last parameter `loggingOn` defines whether the debug logging capabilities shall be enabled or not. In addition to the debugging parameter the Log event can be subscribed, to receive a callback when a log message is available. After each time step the model status can be read, to know whether everything is still working or an error occurred.

The following function can be called after the model was instantiated:

```
Fmi2Status SetupExperiment(bool toleranceDefined, double
   tolerance, double startTime, bool stopTimeDefined, double
    stopTime)
```

With this function the experiment can be setup. The function returns a `Fmi2Status`, which is an enumeration and can be one of the following values to show the status of the model:

- fmi2OK,
- fmi2Warning,
- fmi2Discard,
- fmi2Error,
- fmi2Fatal,
- fmi2Pending

Depending on the status value different actions can be taken. For the use of this master thesis it is only necessary to check whether the status is `fmi2OK`. When the status was `fmi2Warning`, `fmi2Discard`, `fmi2Error` or `fmi2Fatal` they simulation should be stopped and the error investigated. If the status was `fmi2Pending`, then the calculation of the model is not finished and it should be waited for it to finish. Additionally, a timeout could be implemented to only wait for a certain amount of time and if the time is passed, to stop the simulation and to investigate if there is an endless loop or another problem occurring.

The parameters `toleranceDefined` and `tolerance` are different depending on whether the FMU is a Model Exchange or a Co Simulation FMU type. These parameters are for error estimation of the step size during simulation. This can be ignored as the step size will be controlled by the adapter that starts the FMU. The start time and stop time parameters can be set in order to let the model know whether how much memory needs to be allocated. When the `stopTimeDefined` is set to `fmi2True` the stop time is fixed and the model will show a `fmi2Error` when the simulation runs past the stop time.

As the stop time is not known beforehand of the simulation in VECTO the parameter will be set to false and the stop time parameter is ignored.

After the `SetupExperiment` function the model can enter the initialization mode with the function `fmi2EnterInitializationMode` and exit it with `fmi2ExitInitializationMode`. In this mode the initial values for some inputs can be set or calculated.

After exiting the initialization mode, the rest of the inputs can be set for the first time step. The input data can be set for with the mentioned function `SetReal`. After all the input signals are set, the time step can be executed with the following function:

```
Fmi2Status DoStep(double currentCommunicationPoint, double
    communicationStepSize, bool
    noSetFmuStatePriorToCurrentPoint)
```

The parameter `currentCommunicationPoint` is the current time and starts with zero. It is increased with each time step by 0,1. The parameter `communicationStepSize` is 0,1 in every step for the VECTO simulation. The parameter `noSetFmuStatePriorToCurrentPoint` is set to true when the function `fmi2SetFMUState` is no longer called in this time step. The state of the FMU can be saved with the function `fmi2GetFMUState`. It saves the state of FMU and can be set again. This helps when an error occurred in the FMU or when an older state is needed. This functionality is not needed for the simulation in VECTO but could be used in the future when the state in the model may be more important.

After the `doStep` function finished, the status can be observed, which shows whether the step was successful. When the `doStep` function is run asynchronously the status can either be observed with the function `fmi2GetStatus` or by subscribing to the callback function `fmi2StepFinished`. When the step is finished the outputs can be read with the function `fmi2GetReal` and passed back to the `SimulinkModelShiftStrategy` functions that called the step function of the adapter.

The return values of the model can only be integers, therefore in the function `shiftRequired` the value has to be cast to a boolean. This is done by a simple If-statement to check whether the output is zero and therefore the boolean is true or if the output is one, then the boolean is false.

All of this is done in the FMI Adapter in the VECTO project. The class in which everything happens is the `FMUConnector` class and is being called by the `SimulinkModelShiftStrategy` class that implements the AMT shift strategy. The new shift strategy can be exchanged for the AMT shift strategy. This is done for testing of this thesis manually, but in the future the ability to set it in the GUI of VECTO can be added.

In VECTO multiple runs are executed in parallel and therefore the FMU is run multiple times at the same time as well. But the FMU DLL has to be extracted from the FMU

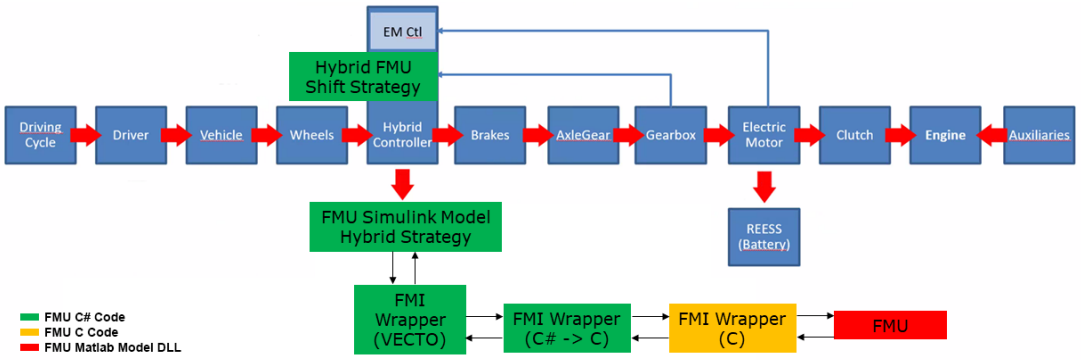


**Figure 6.4:** Structure of Hybrid VECTO and the FMU Matlab Simulink model

ZIP file and copied to another place in order to load the DLL and instantiate the FMU. Therefore, the DLL is copied to the current logged in Windows users temporary folder.

## 6.2 OEM Hybrid Strategy

After showing the implementation of the generic shift strategy of VECTO in Simulink and loading it as an FMU in VECTO works as expected, the next step was to use a hybrid strategy Simulink model of an OEM in VECTO. The model is from an OEM and is already available as a Simulink model. The goal was to use the strategy and show that a hybrid strategy can be made compatible with VECTO. A new adapter has to be written for VECTO, as the new hybrid version of VECTO and the used shift and hybrid strategies are connected with each other. The strategy of the OEM is only the hybrid strategy, therefore the shift strategy in VECTO has to be adapted to work with an external hybrid strategy.

The structure of Hybrid VECTO and the added classes are shown in Figure 6.4. Instead of the VECTO Hybrid Strategy, the FMU Simulink Model Hybrid Strategy forwards the requests to the FMI Wrapper and consequently forwards the request to the Simulink model exported to an FMU. The Hybrid FMU Shift Strategy is an adapted AMT Shift Strategy as the OEM Hybrid Strategy does not have a Shift Strategy, but a Shift Strategy is necessary for the Simulation and is defined in the Hybrid Controller instead of the Hybrid Shift Strategy.

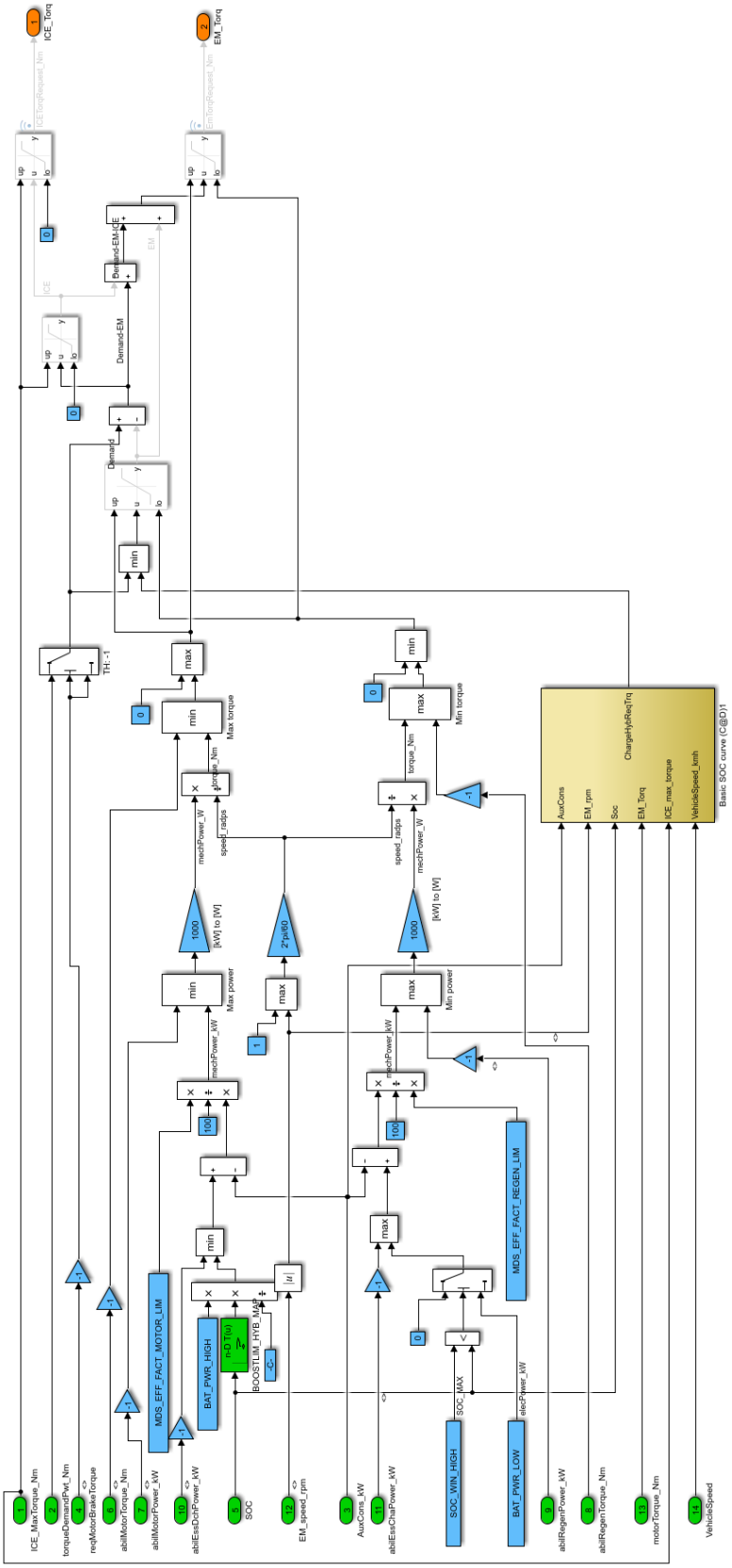


**Figure 6.5:** Matlab Simulink model of the OEM Hybrid Strategy

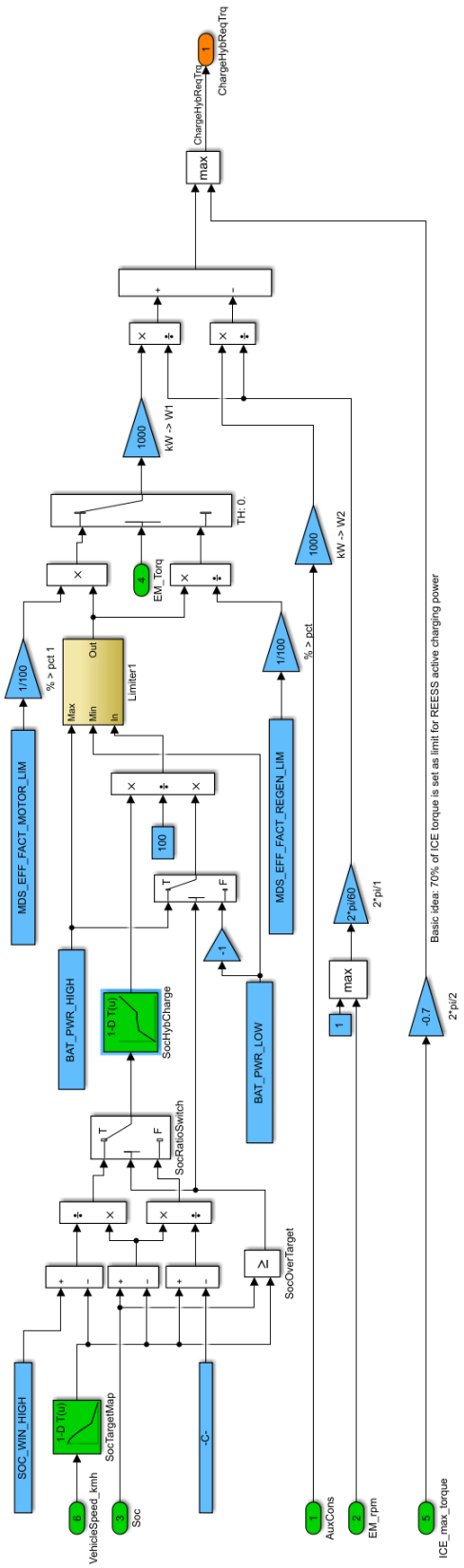


**Figure 6.6:** Matlab Simulink model of the OEM Hybrid Strategy SOC curve

### 6.2.1 Explanation of OEM Hybrid Strategy

The OEM hybrid strategy decides how much torque the electric motor and how much the internal combustion engine provide, which could also be 100% torque from the ICE and 100% from the electric motor. This mainly depends on the charging level of the battery. If the battery is full the vehicle will be driven just by the electric motor, but when the battery is empty the hybrid strategy will let the ICE do all the work and will even use additional torque in for charging the battery by recuperating the electric motor. The model in Simulink can be seen in Figure 6.5. In addition to the model, a small script with some parameters for the model was provided:

```
% Battery power limitation
BAT_PWR_HIGH = 120;      % [kW]
BAT_PWR_LOW = -120;      % [kW]
### % Battery Soc Window ###
SOC_WIN_LOW = 25;
SOC_WIN_HIGH = 60;
### % Electric moteur efficiency ###
MDS_EFF_FACT_MOTOR_LIM = 96.0;            % [%] Electric
   motor efficiency factor at motor limit
MDS_EFF_FACT_REGEN_LIM = 96.0;            % [%] Electric
   motor efficiency factor at regen limit
### % Boost limitation map depending of the SOC ###
BOOSTLIM_VECTO_HYB_MAP.x=[25.00     27.0    28.0    30.0
   31.0    32.0    33.0    34.0    36.0];  % [%] SOC
BOOSTLIM_VECTO_HYB_MAP.y=[0.00      10.0    15.0    30.00
   50.0    70.0    90.0    100.00  100.00];
### % Charge while driving limitation map ###
%CHARGE_REQ_LIMIT_MAP.x = ems.mt_MAX_TORQUE_HIGH_MAP_x; % [
   rpm] # Max allowed charge request torque on the electric
   motor.
### % Soc target map ###
SOCTARGET_VECTO_MAP.x = [ 0.0       10.0    20.0    30.0
   35.0    40.0    45.0    50.0    100.0 ;
                                      % [km/h]
SOCTARGET_VECTO_MAP.y = [37.50      37.00   36.01   34.0
   33.00   32.0    31.00   30.0    28.0];
### % Charge while driving percentage map depending of the
   Soc ratio ###
SOCCURVE_HYB_VECTO_MAP.x = [-1.0000 -0.5    0.0     0.3
          1.0000]; %SOC-ratio
SOCCURVE_HYB_VECTO_MAP.y = [-100.00 -10.0   0.0     0.00
   100.00]; %PWR-perc
```

Furthermore, a small submodel can be seen in Figure 6.6. The basic concept of the OEM model is that depending on the State of Charge (SoC) of the battery and limited by parameters such as maximum electric motor torque, maximum charging and discharging power, maximum ICE torque, auxiliaries power, electric motor speed the needed torque demand or braking torque. Depending on the parameters, such as the battery conditions or the speed of the vehicle, the calculation of the torque provided is influenced. Due to these parameters and calculations, the maximum torque that can be provided and the maximum recuperating torque that is possible is calculated. Between these two limits the final torque is decided on.

Some signals of the OEM model are different than in VECTO. VECTO calculates some signals with positive values but which negative in the OEM model, therefore these values had to be changed for the model to work with VECTO.

- $reqMotorBrakeTorque$
- $abilMotorTorque\_Nm$
- $abilMotorPower\_kW$
- $abilEssDchPower\_kW$
- $EM\_speed\_rpm$

These values had to be multiplied by -1 to make them positive, but for the parameter $EM\_speed\_rpm$ the absolute value of it had to be calculated, as the parameter can be both positive and negative in VECTO.

All the parameters in the model need a unit. These units need to be matched by VECTO as well. If the units between VECTO and the model do not match, the calculation will be wrong. The following table shows the mapping between the parameters in VECTO and in the OEM model:

The maximum torque of the ICE is passed as the parameter `ICE_MaxTorque_Nm`. In VECTO this parameter relates to the parameter
`DynamicFullLoadTorque` in the engine component. The `IntertiaTorqueLoss` and the `AuxTorqueDemand` had to be subtracted from this value. This parameter is used in the model to known how much the ICE can handle at maximum.

The `torqueDemandPwt_Nm` is the parameter that tells the model how much torque is currently needed. The model has to decide how much of this torque should be provided by either the ICE or the electric motor.

The parameter `AuxCons_kW` is the power demand of the auxiliaries that needs to be included in the calculation of the available power for the electric motor.The parameter `abilEssDchPower_kW` is the maximum discharge power of the battery system. In VECTO this parameter is found in the electric system, specifically the Rechargeable

| VECTO | OEM Model |
|---|---|
| ICE_MaxTorque_Nm | Engine.DynamicFullLoadTorque - Engine.IntertiaTorqueLoss - Engine.AuxTorqueDemand |
| torqueDemandPwt_Nm | Gearbox.InTorque |
| AuxCons_kW | Engine.AuxiliariesPowerDemand |
| abilEssDchPower_kW | ElectricSystem.RESSResponse. MaxDischargePower |
| abilEssRegenPower_kW | ElectricSystem.RESSResponse. MaxChargePower |
| SOC | response.ElectricSystem.RESSResponse. StateOfCharge |
| abilMotorTorque_Nm | response.ElectricMotor.MaxDriveTorque |
| abilMotorPower_kW | ElectricMotor.MaxDriveTorque * ElectricMotor.AngularVelocity |
| motorSpeed_rpm | Engine.EngineSpeed |
| abilRegenPower_kW | ElectricMotor.MaxRecuperationTorqueEm * ElectricMotor.AngularVelocity |
| abilRegenTorque_Nm | ElectricMotor.MaxRecuperationTorqueEm |
| EM_speed_rpm | ElectricMotor.EmSpeed |
| motorTorque_Nm | DataBus.ElectricMotorInfo(VectoCommon. InputData.PowertrainPosition.HybridP2) as ElectricMotor).CurrentState.EmTorq ueMap |
| VehicleSpeed_kmh | DataBus.VehicleInfo.VehicleSpeed |

**Table 6.1:** VECTO and OEM model signals mapping

Electric Energy Storage System (REESS). In contrast, the parameter `abilEssRegenPower_kW` is the opposite, it is the maximum regeneration power the electric system can provide to charge the battery. The `SOC` parameter is the state of charge of the battery. The `abilMotorTorque_Nm` and `abilMotorPower_kW` parameters specify the maximum torque and power of the electric motor. The `motorSpeed_rpm` parameter is the speed of the combustion engine. The `abilRegenPower_kW` and the `abilRegenTorque_Nm` parameters are how much power and torque is possible to charge the battery. The `EM_speed_rpm` is the speed of the electric motor. The `motorTorque_Nm` is how much torque the electric motor currently provides. The `VehicleSpeed_kmh` is the current speed of the vehicle.

### 6.2.2 Implementation of FMU Adapter in Hybrid VECTO

The FMU adapter in the hybrid VECTO version is very similar to the version that was used for the current official VECTO version. The class `FMUConnector` handles all communication with the FMU, initializing the FMU and terminating it. The inputs are the signals that have been described above. The output signals are the torque of the electric motor and the torque of the combustion engine. For this implementation only the signals the OEM models needs are passed to the model. In future versions the passed signals need to be standardized. All available signals required by a hybrid strategy could need should be provided. It will be the task of the model to use the signals in the correct format it needs. Currently, unit and positive to negative calculations are done in VECTO and not in the Simulink model in order to retain the OEM model as how it was provided.

### 6.2.3 Adapted VECTO Hybrid and Shift Strategy for the OEM Strategy

In the hybrid VECTO version the shift and hybrid strategies are connected with each other. As this is not the case with the OEM strategy, as it consists only of the hybrid strategy, the AMT shift strategy had to be adapted to work with the OEM Strategy. Additional code in the hybrid strategy of VECTO was needed to handle special cases and let the shift strategy decide when to shift. To get all the necessary parameters for the OEM strategy, a dry run has to be executed before. This is due that the OEM strategy uses a forwards calculating approach instead of the backward calculation of VECTO.

The hybrid controller in VECTO instantiates the shift strategy and has a connection to the hybrid strategy. The `HybridFMUShiftStrategy` implements the `HybridCtlShiftStrategy` but is based on the `AMTShiftStrategy` with only some minor customizations to work with the hybrid strategy. The hybrid controller gets a request

from the previous component and sends the request to the hybrid strategy. The controller gets the response from the hybrid strategy, triggers a gearshift if necessary and forwards the request to the next component.

The main changes in the `HybridFMUShiftStrategy` were done with the conditiona that the combustion engine is turned off and electric motor operates only. For this mode the gear has to be shifted correctly, in case the combustion engine is getting turned back on. The speed for the combustion engine should not be too high or too low when it is turned back on, therefore it is still necessary to shift gears while the combustion engine is turned off. It is necessary for the electectric motor as well to shift gears, as the electric motor has a maximum speed it is capable off.

The `FMUSimulinkModelHybridStrategy` is the new hybrid strategy to work with the OEM hybrid strategy. When a request is executed it checks whether a gear is currently engaged or not. It also checks which driving action is currently happening and whether there is a saved solution of a previous dry run with the same driving action. When the driving action is different from the saved one, the dry run solution is deleted and a new strategy configuration is calculated. A dry run solution cannot be used for a different driving action. When the driving action is the same and a saved dry run solution exists, the same strategy configuration of the electric motor and ICE is returned. This is the same strategy approach as used in the VECTO `HybridStrategy`.

When there is no dry run solution saved and the gear is engaged, a dry run is requested. When a dry run is executed it wants a configuration of the electric motor and ICE to use for the dry run. A configuration with the electric motor turned off and the combustion engine turned on is used. This dry run request is needed to gather all the parameters for the OEM strategy. Some parameters are already available, but the hybrid controller is located between the wheels and the brakes in the VECTO powertrain. All the parameters after the hybrid controller component are not yet calculated but need to be available for the OEM strategy. Therefore, a dry run is executed with the current parameters and all the data of every component for this time step are calculated on the simplified test powertrain. With the response of the dry run, the request to the FMU OEM hybrid strategy can be sent.

When the data are returned from the OEM model, a special case needs to be checked. The OEM hybrid strategy only returns how much torque the electric motor and how much the ICE torque should be doing in this time step. When the combustion engine is turned back on and should provide torque and the vehicle is not stopped it needs to be checked whether the correct gear is currently engaged. This can happen when in the previous time step the combustion engine is turned off and is suddenly turned on, but the speed is too high for the engine and cannot provide the required torque. Therefore, it needs to be shifted to the next gear for turning the combustion engine back on. This special case is checked in the shift strategy, as explained above, but needs to be checked at this point as well.

After the two parameters of the OEM hybrid strategy are returned, it needs to be checked whether the electric motor or ICE should be turned off or turned back on. When the current driving action is `Halt` or the OEM hybrid strategy returns that the ICE and electric motor both provide zero torque or only the electric motor provides torque. In this cases the combustion engine should be turned off. The electric motor should also be turned off when the torque returned for the electric motor is zero or no gear is engaged. When no gear is engaged the strategy configuration is that the combustion engine is turned on, but the electric motor is turned off.

In order to test the OEM hybrid strategy and implementation of the `HybridFMUShiftStrategy` and the `FMUSimulinkModelHybridStrategy`, some test cases were created that are based on existing cases. The test cases consist of constant speed drives with 30, 50 and 80 km/h. As well as acceleration test cases, called drive off, with 30 and 80 km/h and a stand still test case with 50 km/h. These test cases have variations with different battery levels and street slopes. After these simple test cases worked, the drive cycle test were performed: Coach, Construction, Inter Urban, Long Haul, Regional Delivery, Sub Urban, Urban and Urban Delivery. With all of these test cases working, the OEM hybrid strategy and the implementation proved to work for the majority of drive cases. It is still possible that some special cases of driving or battery and vehicle configurations are not yet covered with this implementation and need to be taken care of in the future.

# 7 Evaluation

In the first section of this chapter the `AMTShiftStrategy` and its implementation as an FMU is evaluated and furthermore it is analyzed whether there are any differences to the VECTO implementation in terms of result, calculation speed and which steps are needed in order that OEMs can use their own models in the future.

In the second section of this chapter the OEM Hybrid Strategy used as an FMU is evaluated in terms of the result and the speed differences as well. These will be more extensive than the `AMTShiftStrategy` in the first section, as the OEM Hybrid Strategy is completely different from the VECTO Hybrid Strategy in both results and calculation speed. It will require more work to ensure compatibility with manufacturer models as the Hybrid Strategy closely interacts with the Shift Strategy in VECTO.

## 7.1 FMU with VECTO AMT Shift Strategy

The results of the VECTO `AMTShiftStrategy` are compared with the implementation of the `AMTShiftStrategy` in Simulink and exported as an FMU. There are 2 versions of implementations, the one with static memory for arrays used in Matlab version R2016b and the dynamic memory implementation for arrays used in Matlab version R2017a and above.

### 7.1.1 Result Differences

The results do not differ from each other as the same code of the VECTO `AMTShiftStrategy` was replicated in the Simulink model. The only differences that could affect some digits after the decimal point would be the use of the bilinear interpolation instead of a Delaunay map, and when Matlab and C# calculations are implemented differently. But as only basic operations are used in the calculations they should be the same. Differences in testing are due to different vehicle parameters, as the internal values of some parameters such as the `Engine.StartSpeed` are different from the JSON files, that were provided for the test. Therefore the results of the offical VECTO and the Matlab Simulink model with the AMT Shift Strategy are different. However it shows that the parameters can have high implications in the results of the simulation as can be seen in Figure 7.1. In the first drive off, acceleration of the vehicle of the VECTO

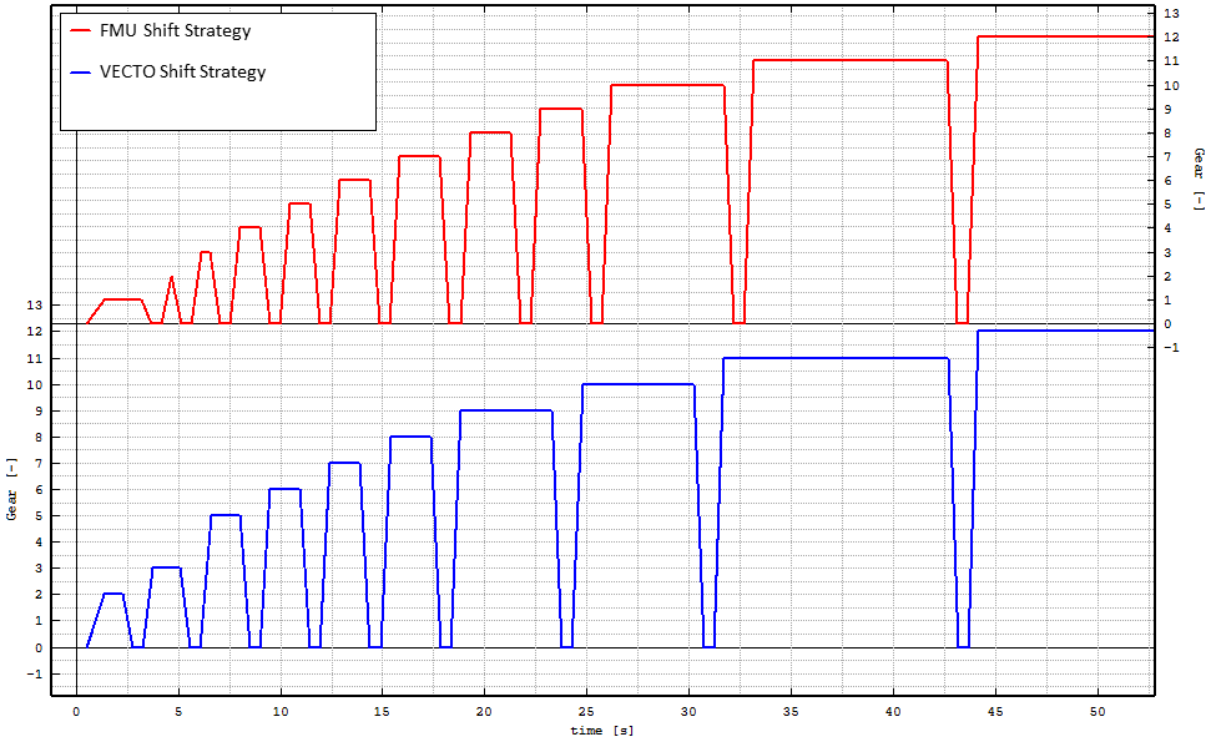


**Figure 7.1:** Comparison of current gears engaged between VECTO and FMU Simulink

version (blue line) shifts instantly into gear two, while the FMU Simulink version (red line) shifts only into gear one. This is caused by a different start speed. This difference will influence the whole simulation from this starting point. The first brake action has different gear shifts too. As can be seen in the Figure 7.2, the VECTO version (blue line) will shift down two gears at once to gear six, while the FMU Simulink version (red line) line will only shift down one gear by one until gear eight.

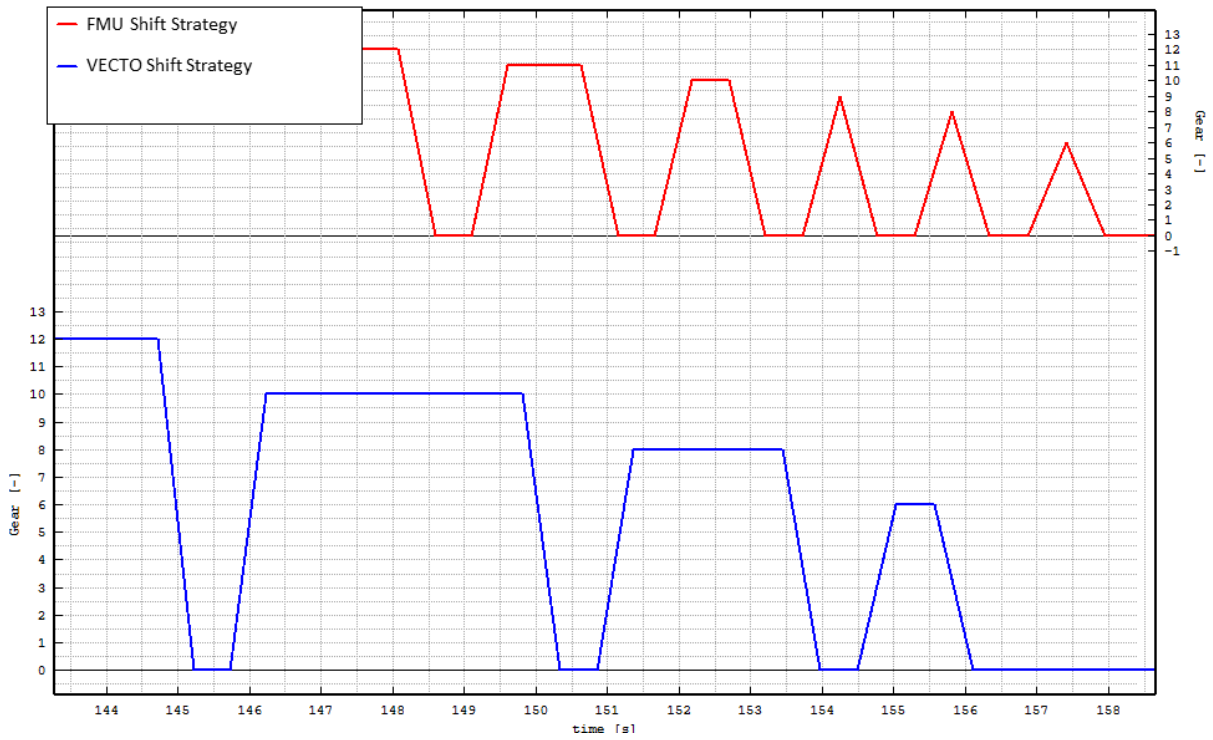


**Figure 7.2:** Comparison of current gears engaged between VECTO and FMU Simulink

### 7.1.2 Calculation Time

The difference in calculation time of the simulations may not be as important. Yet, it still shows how the implementations function differently and whether these will lead to time losses during the simulation. These time losses can occur when data need to be loaded and sent via a communication channel, or when an algorithm runs longer than other implementations with more efficient algorithms.

The simulation duration for the `AMTShiftStrategy` can be differentiated between the VECTO `AMTShiftStrategy` implementation, the Matlab Simulink implementation of the `AMTShiftStrategy` with dynamic memory enabled for arrays, and the implementation in Matlab Simulink with only static memory for the arrays.

The VECTO `AMTShiftStrategy` for all 10 cycle runs took 34 seconds, while the Simulink `AMTShiftStrategy` took 89 seconds with dynamic memory and 57 seconds with static memory.

The VECTO `AMTShiftStrategy` is the fastest strategy, which is obvious as the whole simulation happens in C# and no data needs to be transferred over a communication channel. The Matlab Simulink implementation of the `AMTShiftStrategy` needs more time as the data are passed to the FMU, and when the FMU simulation is finished the result is passed back to VECTO. The Matlab Simulink version with static memory is

**Table 7.1:** VECTO and OEM model signals mapping

| Implementation | Speed |
|---|---|
| VECTO AMTShiftStrategy | 34s |
| Matlab Simulink AMTShiftStrategy (dynamic memory) | 89s |
| Matlab Simulink AMTShiftStrategy (static memory) | 57s |

faster, because the memory can be acquired in the beginning of the simulation and does not change in size anymore. Dynamic memory allocation is slower as this is done in the beginning and expanded when necessary. The allocation of additional memory takes some execution time and slows down the application. Although static memory allows a faster simulation, for avoiding issues when accessing the data it is still preferable to use dynamic memory. The calculation times can vary depending on the computer that is used for the simulations, but the differences between the three versions will stay the same.

### 7.1.3 Further Work to Ensure Compatibility with Manufacturer Models

Parts of the implementation have to be standardized in order to ensure that multiple OEMs can use their own version of a Shift Strategy for the simulation of VECTO. The most important thing to standardize and to stay fixed for OEMs to work with are the signals that are passed to the model during simulation. The in-ports in Matlab Simulink need to have the correct names in order that these signals can be read in C#, that means these names have to match. The easiest solution would be to create a Matlab Simulink Model only with in- and out-ports with the standardized names as a template for the OEMs to use. The OEMs can import their model as a Subsystem in this template and connect the signals they need. This could also be paired with versioning, when new signals are added or needed by OEMs. The versions need to be matched between the template and OEM model and back compatibility needs be ensured for them. In addition to the signals the vehicle data JSON file, which is loaded in the beginning of the simulation in the model, needs to be standardized too. The changes that were needed in order to correctly load the JSON file were documented in Chapter 6.1.1.

In the current implementation of the adapter for the Matlab Simulink `AMTShiftStrategy` model as an FMU the 3 functions, that are public in the `AMTShiftStrategy`, are just passed on to be handled by the Matlab Simulink model. This could be simplified by passing on only one function to the model for easier handling, but this would need the `AMTShiftStrategy` to be refactored to work.

Another problem is debugging when errors occur in Matlab Simulink model. Currently, it is not easy to debug the Matlab Simulink model when an error occurs during the simulation. In this case, Visual Studio only displays an error for the `doStep` function in the FMU adapter. The easiest way to debug is to check at which `absTime` the error happened, to export the parameters at this time step and to set the parameters of this time step as input in Matlab Simulink. The next step is to debug in Matlab Simulink where the error occurs in the code. This could be done automatically by saving the parameters and the time step in case of an error, and export them to a CSV file to be used for Matlab Simulink. [72]

Another possibility is that Matlab Simulink exports the debug symbols into a .pdb file when compiling the model. This file can then be added to Visual Studio. With the debug symbols the debugger knows the code lines and can show a stack trace of the error. For this master thesis it was only tested whether the debug symbols file is created, but not how they can be imported into Visual Studio. Due to the process of loading an FMU in C#, it is rather complicated to correctly add these debug symbols to the Visual Studio project. This would need some additional investigations on how to load debug symbols for C code DLLs.

The debug symbols can be exported from Matlab Simulink with the following compiler flags:

```
coder.updateBuildInfo('addCompileFlags', '/Zi');
coder.updateBuildInfo('addLinkFlags', '/DEBUG');
```

The .pdb files can be found at:

```
<currentFolder>\slprj\_sfprj\<model>\_self\sfun\src\
```

## 7.2 OEM Hybrid Strategy

The results between the VECTO Hybrid Strategy and the OEM Hybrid Strategy are very different, as the OEM Strategy is much simpler than the VECTO strategy. The differences between the two are investigated into much more depth in the following sections, especially how the two versions are shifting gears and how much torque the electric engine and the ICE provide.

### 7.2.1 Result Differences

While the OEM Hybrid Strategy is more simplistic, the VECTO Hybrid Strategy has a few techniques in order to improve certain scenarios, for example shifting up and skipping gears when accelerating. This can be observed quite good in the following Figure 7.3.

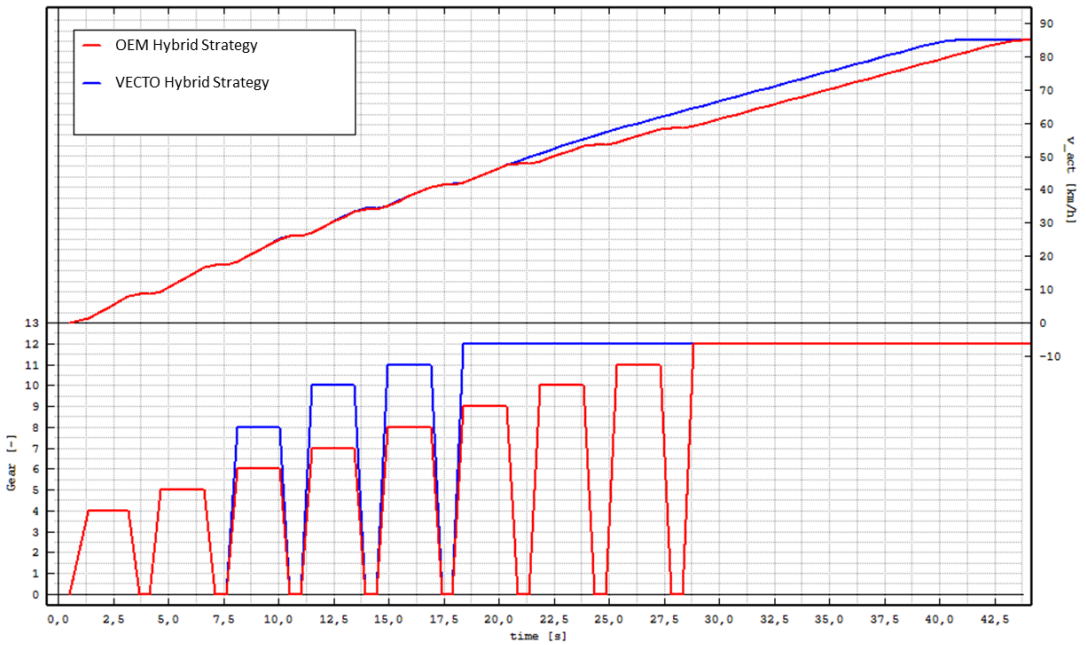


**Figure 7.3:** Comparison of current gears engaged between VECTO and OEM Hybrid Strategy

The blue line in the bottom diagram of Figure 7.3 shows the currently engaged gear of the VECTO Hybrid Strategy, whereas the red line shows that of the OEM Hybrid Strategy in the cycle run Long Haul. The data are taken from the tests that were written in order to test the functionality. The tests use a generic vehicle with generic parameters, which are not optimized. However even without the optimized parameters the differences between these two strategies can be observed.

The VECTO Hybrid Strategy skips gears in the acceleration process whereas the OEM Hybrid Strategy does not, it only shifts from one gear to the next. This is one of the scenarios in VECTO that improves the overall results of the run. The option to skip gears is implemented in the OEM Hybrid Strategy, but only when the speed is too high for the engine speed. The VECTO Hybrid Strategy checks whether it can skip a gear even when the speed is not yet too high for the engine, but will probably be too high in the next time steps. At the top half of the Figure 7.3 the actual vehicle speed in km/h can be seen, where again the red line is of the VECTO Hybrid Strategy and the blue line is of the OEM Hybrid Strategy. The VECTO Hybrid Strategy calculates a faster acceleration due to the more efficient gear shifting.

The difference between the two implementations how much torque the electric motor should provide or how much recuperaton is done to charge the battery can be seen in Figure 7.4. The torque signal of the electric motor is shown in blue for the VECTO Hybrid Strategy and in red line for the OEM Hybrid Strategy. When the torque is

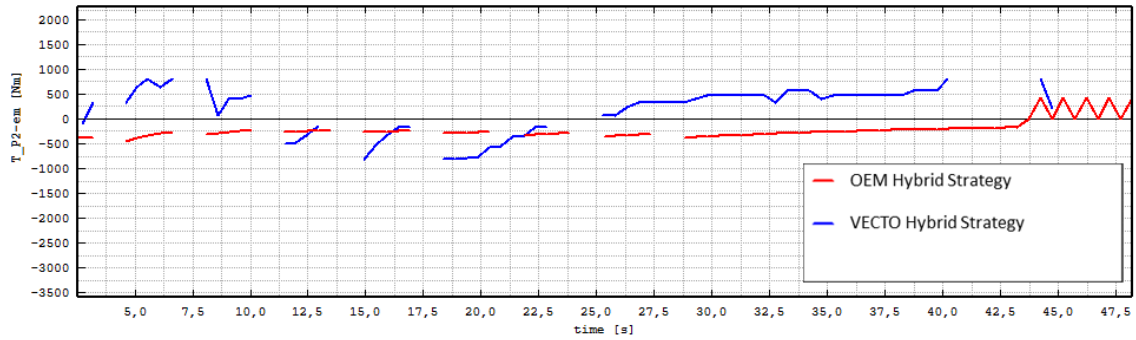


**Figure 7.4:** Comparison of the electric motor torque between VECTO and OEM Hybrid Strategy

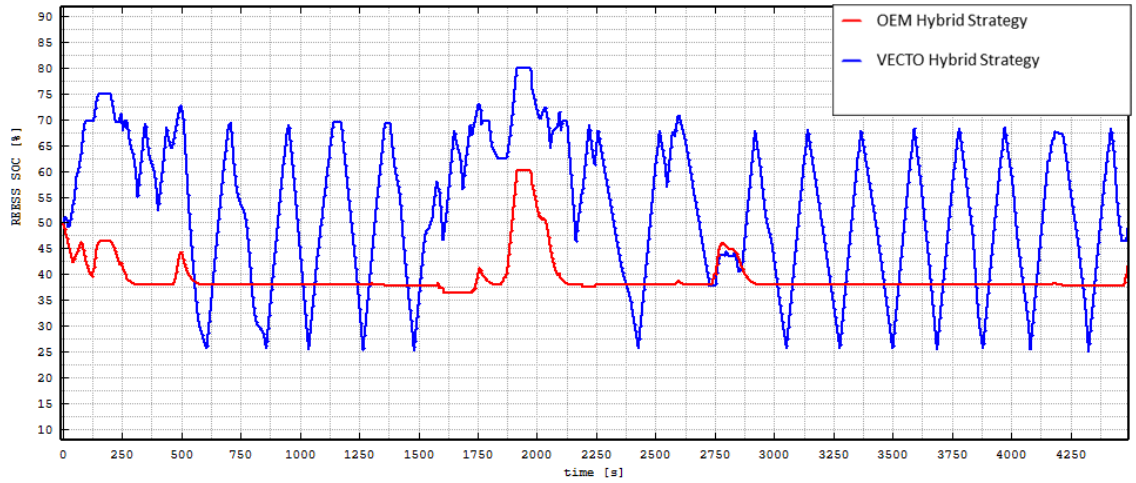


**Figure 7.5:** Comparison of Battery SOC between VECTO and OEM Hybrid Strategy

positive the electric motor recuperates, and when its negative it does part of the work to propel the vehicle forward. The OEM Hybrid Strategy uses the electric motor in the beginning to accelerate but when driving at a constant speed it switches the electric motor between recuperation and acceleration as it goes below and above a certain SOC level. This does not happen all the time, as it switches to the ICE for driving at constant speed at certain points. It depends on the slope of the street as well. The VECTO Hybrid Strategy is set to keep a higher SOC level generally. When it switches to the electric motor it will keep it turned on for a certain amount of time until it needs to recuperate again. This can be seen in Figure 7.5, where the blue line is the VECTO Hybrid Strategy battery SOC levels and the red line is the one of the OEM Hybrid Strategy.

In the following Table 7.2 the differences of the results of the cycle Long Haul run are shown. The VECTO Hybrid Strategy overall time was a bit shorter and therefore the average speed was a bit higher too. But the value "FC-Map", which is the average fuel consumption ("before all corrections, interpolated from the Fuel Map, based on the torque and engine speed" described in the VECTO User Manual), is higher for the VECTO Hybrid Strategy than for the OEM Hybrid Strategy. This can be explained by the higher number of ICE starts in the VECTO Hybrid Strategy. On the other hand the ICE is turned off more often for the VECTO Hybrid Strategy than for the OEM Hybrid Strategy. The OEM Hybrid Strategy used the electric motor much less often and let the ICE do most of the work. It is assumed that the higher average fuel consumption of the VECTO Hybrid Strategy is caused by the higher number of ICE starts and by a recurring cycle of charging the battery SOC up to 75% and then using the electric motor until the battery SOC hits 25%. The differences of the battery SOC level traces of the VECTO Hybrid Strategy (blue line) and the OEM Hybrid Strategy (red line) can be seen in Figure 7.5. The OEM Hybrid Strategy is always set to maintain a SOC level of about 38%, except for a few cases where the vehicle could recuperate more and charge the SOC level up to 60%. For acurate comparison of the results it would be needed to correct them by the SOC level. It is part of the VECTO Hybrid Strategy to always charge the battery and then to discharge it. This keeps the battery more healthy in comparison of keeping a constant SOC level.

In the table above only the cycle run Long Haul was examined in more detail, but similar behavior was observed in the other cycle runs.

### 7.2.2 Calculation Time

The difference in calculation between the two strategies is much larger than the differences of the individual implementations of the AMT Shift Strategy, even with the added time it takes to pass the data to the FMU and receiving the results back. This can be attributed to the implementation of the VECTO Hybrid Strategy which triggers

**Table 7.2:** VECTO and OEM Hybrid Strategy Result Differences in the Long Haul cycle run

| | OEM | VECTO |
|---|---|---|
| Simulated Driving Time [s] | 4514,45 | 4512,16 |
| Average vehicle speed [km/h] | 79,89 | 79,93 |
| FC-Map [g/h] | 11406,54 | 12106,45 |
| FC-Map [g/km] | 142,78 | 151,46 |
| Number of gearshifts [-] | 48 | 41 |
| ICE starts [-] | 27 | 51 |
| ICE off time share [%] | 12,42 | 52,07 |
| EM off time share [%] | 78,30 | 9,50 |

many dry runs to test out different configurations for evaluation of ICE and electric motor torque and when to shift gears. The time it took for the test of the Long Haul cycle run was 46.2 seconds for the OEM Hybrid Strategy. The duration for the same run in the VECTO Hybrid Strategy was 504 seconds. These time durations may vary depending on the computers being used for the simulations, but the differences between the two implementations should stay the same.

### 7.2.3 Further Work to Ensure Compatibility with Manufacturer Models

Letting OEMs use their own Hybrid Strategies is much more complicated than with the AMT Shift Strategy. The AMT Shift Strategy is much simpler and only contains data about when to shift up and down and all associated signals with it. In contrast, the Hybrid Strategy works closely with the Shift Strategy in the Hybrid version of VECTO. Due to this, the Hybrid Strategy had to be adapted to work with a modified version of the AMT Shift Strategy to achieve compatibility with the OEM Hybrid Strategy with VECTO. This means the OEM Hybrid Strategy is called by the modified Hybrid Strategy in VECTO, which in turn communicates with a modified AMT Shift Strategy. One option is that a Hybrid Strategy with a modified Shift Strategy, like the one adapted for this thesis, will be provided for the OEMs so that they can use their own Hybrid Strategy. The other option is that the OEMs also have to include a Shift Strategy to their Hybrid Strategy to work with VECTO. This would complicate the interface as well, as two components would be controlled by a manufacturer. Also more data, which are calculated in each component of Vecto, may be needed in the DataBus. Either version on how an OEM can co-simulate with VECTO should work, but both need a standardized and robust connection point to the OEM model.

# 8 Conclusion

The intention of this thesis was to explore the possibilities to connect a Matlab Simulink model to VECTO which is written in C#. The "Matlab Simulink for Co-Simulation" chapter deals with the different technologies on how to exchange data between C# and Matlab Simulink. All of them should work, but they all have their advantages and disadvantages. The decision to export the model was due to the fact that it was better not to run Matlab at the same time, as using Matlab Simulink would apply some restrictions, that a certain version of Matlab Simulink would be needed. This could be an issue as Matlab is already a few years old and companies probably a have different version available and would not like to change it due to licensing and costs. It would not be feasible to restrict VECTO to only work with one Matlab version or increasing the development effort to support multiple Matlab versions. Therefore, the option to use code generation was the best and most promising option. In addition to code generating the model there are a few options on how to use the code in VECTO during the simulation. One possibility is to load the full source code of the model dynamically, but this would restrict the OEMs to share their code and their intellectual property. Therefore, the code needed to be compiled into a Shared Library. This still left the question on how to interact with the model as a Shared Library. The FMI standard provides such an interface, which enables interaction with Matlab Simulink models during simulation and provides an open source approach to export models as FMUs. The only disadvantage of using a code generated model is the reduced feature-set provided by Matlab Simulink. Some parts of the code like loading data, had to be taken care of especially to ensure that they work properly.

With the decision on which technology to use to interact with Matlab Simulink, the VECTO side had to handle the loading of the FMU and start of the simulation. An existing project on how to interact with an FMU was used, which worked very good and didn't show any problems during the development of this thesis. The AMT Shift Strategy was the first part to test the connection between a Matlab Simulink model and VECTO. The AMT Shift Strategy was reimplemented in Matlab Simulink to see whether everything would work and to identify differences in case of failures in the implementation. The implementation of the AMT Shift Strategy had some challenges, as some parts could not be reimplemented the same way in the Matlab scripting language as in C# due to less available out-of-the-box functions of Matlab Simulink. A new Shift Strategy had to be added to VECTO based on the AMT Shift Strategy, but the

function calls to the Shift Strategy was then be forwarded to the FMU and the results were passed back. This worked quite well, but the only problem were errors occurring in the FMU, as it was not easy to debug them. The simplest way was to check the time when the error occurred, save the current time step and parameters, to test the same time step and parameters in Matlab Simulink and to step there through the code to find the error. It should be possible to add debugging functionalities to C# with the available C code and debugging symbols of the FMU which are created when setting the model code generation objective to debugging and additional debug build flags.

The first part was to prove that the simulation works with the VECTO AMT Shift Strategy and the offical version of VECTO. The next part was to use a model of an OEM, the new hybrid version of VECTO and to use them both together. The simple OEM strategy looks complicated in Matlab Simulink, but is much more simple than the VECTO Hybrid Strategy, which proved to be an advantage when investigating errors during development, as it was possible to look up how the OEM Hybrid Strategy calculated the torque of the ICE and electric motor. The Shift Strategy had to be adapted in VECTO in order to work with the OEM model. After the first simple tests with the OEM Hybrid Strategy passed only some special cases remained that happened in the full cycle runs.

In conclusion, the implementation to export a model out of Matlab Simulink with the FMI standard into an FMU and loading it into VECTO is a very promising and easy solution to use for interaction of VECTO with external software. It shows that it is easy to pass data during simulation to the model and to receive results with very low overhead in execution time. It only depends on the duration that the model needs to calculate and to send back the results. The loading of the vehicle information data can be done by exporting it from VECTO into a JSON file and to load this file once at the beginning of the simulation. Sending the data during simulation or in the first time steps via the in-ports of the model is too restricted for large data that are included in the vehicle information, such as the full load curves of the engine or the shift polygons of the gearbox for each gear. The FMU of the models of the manufacturers can be exchanged as well while the Intellectual Property of their strategy is still protected.

When the VECTO simulation works with an external manufacturer model, as is shown in this thesis, it needs to be investigated in which dimensions it should be possible. The problem when exchanging a component of VECTO with that of an OEM is in which dimension the component is able to influence the simulation. The Shift Strategy can only influence gear shifting, in detail, whether to shift early, late or to skip gears when accelerating or breaking. The Hybrid Strategy is even more in control of the simulation and the actions of the vehicle. The vehicle parameters for the control of gear shifting and configuration of the ICE and electric motor torque could drastically change the simulation results.

An open problem, that needs to be discussed and researched further is when OEMs get the possibility to use their own components, is whether the code and model they used for certification are used for the vehicle as well. A process to prove that the same strategies for the simulation and real world operation of the component are applied may be necessary.

# Bibliography


[1] P. Sivakumar, B. Vinod, R. S. Sandhya Devi, S. Poorani Nithilavallee: *Model Based Design Approach in Automotive Software and Systems*, International Journal of Applied Engineering Research, 2015

[2] R. Luz: *Simulationsbasierte Methode zur Zertifizierung der CO2 Emissionen von schweren Nutzfahrzeugen*, Graz University of Technology, Austria, 2015

[3] G. Fontaras, R. Luz, K. Anagnostopoulos, D. Savvidis, S. Hausberger, M. Rexeis: *Monitoring CO2 emissions from HDV in Europe - An experimental proof of concept of the proposed methodological approach*, 2014

[4] N. Hill, S. Finnegan, J. Norris, C. Brannigan, D. Wynn, H. Baker, et.al.: *Reduction and Testing of Greenhouse Gas (GHG) Emissions from Heavy Duty Vehicles - LOT1: Strategy*, Final Report to the European Commission – DG Climate Action; Contract N° 070307/2009/548572/SER/C3; 22.2.2011

[5] S. Hausberger, M. Rexeis, A. Kies, L-E. Schulte, H. Steven, R. Verbeek, et.al.: *Reduction and Testing of Greenhouse Gas Emissions from Heavy Duty Vehicles - Lot 2; Development and testing of a certification procedure for CO2 emissions and fuel consumption of HDV*, Final Report; Contract N° 070307/2009/548300/SER/C3; 9.1.2012

[6] R. Luz, M. Rexeis, S. Hausberger, L. Schulte, J. Hammer, H. Steven, R. Verbeek, et.al: *Development and validation of a methodology for monitoring and certification of greenhouse gas emissions from heavy duty vehicles through vehicle simulation*, Final report; Service contract CLIMA.C.2/SER/2012/0004; Report No. I 07/14/Rex EM-I 2012/08 699; 15.05.2014

[7] M. Rexeis, M. Quaritsch, S. Hausberger, G. Silberholz, A. Kies, H. Steven, M. Goschütz, R. Vermeulen: *VECTO tool development: Completion of methodology to simulate Heavy Duty Vehicles' fuel consumption and CO2 emissions Upgrades to the existing version of VECTO and completion of certification methodology to be incorporated into a Commission legislative proposal.*, Final report; Framework contract CLIMA.C.2/FRA/2013/0007; Report No. I 15/17/Rex EM-I 2013/08 1670; 30.10.2017

[8] F. González, M. González, A. Mikkola: *Efficient coupling of multibody software with numerical computing environments and block diagram simulators*, Proceedings of the Nordic MATLAB Conference, 2003

[9] M. Rexeis, M. Röck, M. Quaritsch, S. Hausberger, G. Silberholz: *Further Development of VECTO*, Final Report; Specific contract N° 340201/2018/776882/SER/CLIMA.C.4; Report No. I 16/19/Rex EM-I 17/24 5670; 29.11.2019

[10] G. Silberholz, S. Hausberger: *Feasibility assessment regarding the development of VECTO for hybrid heavy-duty vehicles*, Report; CLIMA.C.4/ETU/2016/0005LV; Report Nr.: I-19/17/Sil EM I-2016/07-679; 20.11.2017

[11] A. Englisch, T. Pfund, D. Reitz, E. Simon, F. Kolb: *Synthesis of various hybrid drive systems*, 2017

[12] C. Luján, E. Martínez, M. Tobar, A. Checa, X. F. Font, T. Breemersch, S. Hausberger, M. Rexeis: *Bodies and trailers - development of CO2 emissions determination* Report; Procedure no: CLIMA/C.4/SER/OC/2018/0005; 4.6.2019

[13] Matlab - A Brief History of MATLAB `https://de.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html` Visited on 02.01.2022

[14] C. Gomes, C. Thule, D. Broman, G. P. Larsen, H. Vangheluwe: *Co-Simulation: A Survey* ACM Computing Surveys, Volume 51, Issue 3, Article No.: 49, pp 1–33, May 2019

[15] J. Bastian, C. Clauß, S. Wolf, P. Schneider: *Master for Co-Simulation Using FMI* Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22, 2011

[16] B.C. Raczkowski, N. Jones, T. Deppen, C. Lucas, et al.: *A MATLAB Simulink Based Co-Simulation Approach for a Vehicle Systems Model Integration Architecture* SAE Int. J. Advances & Curr. Prac. in Mobility 2(3):1150-1159, 2020, 10.3.2020

[17] R. Odegard, Z. Milenkovic, J. Henry, M. Buttacoli: *Model-Based GN&C Simulation and Flight Software Development for Orion Missions beyond LEO* 2014 IEEE Aerospace Conference, 2014, pp. 1-13, 1-8 March 2014

[18] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, A. Andreev: *Automatic code generation from Matlab/Simulink for critical applications* 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE), 2014, pp. 1-6, 4-7 May 2014

[19] Github - Trick Simulation Environment `https://github.com/nasa/trick` Visited on 02.01.2022

[20] Modelon - FMI TOOLBOX FOR MATLAB/SIMULINK https://www.modelon.com/products-services/modelon-deployment-suite/fmi-toolbox/ Visited on 02.01.2022

[21] powersys-solution - Perform co-simulations between Simulink® and other software https://powersys-solutions.com/product/?software=Easy%20FMI%20Add-On%20for%20MATLAB/Simulink Visited on 02.01.2022

[22] Github - FMI Kit for Simulink https://github.com/CATIA-Systems/FMIKit-Simulink Visited on 02.01.2022

[23] Synopsys - FMU SDK https://www.synopsys.com/verification/virtual-prototyping/virtual-ecu/fmu-sdk.html Visited on 02.01.2022

[24] Github - FMU SDK https://github.com/qtronic/fmusdk Visited on 02.01.2022

[25] Github - FmiWrapper https://github.com/Tuebel/FmiWrapper/ Visited on 02.01.2022

[26] Github - cJSON https://github.com/DaveGamble/cJSON Visited on 02.01.2022

[27] Matlab Documentation - Calling MATLAB as COM Automation Server https://www.mathworks.com/help/matlab/call-matlab-com-automation-server.html Visited on 02.01.2022

[28] Matlab Documentation - MATLAB COM Automation Server Interface https://www.mathworks.com/help/matlab/matlab_external/matlab-com-automation-server-interface.html Visited on 02.01.2022

[29] Matlab Documentation - COM Objects in MATLAB https://www.mathworks.com/help/matlab/using-com-objects-in-matlab.html Visited on 02.01.2022

[30] Matlab Documentation - Choosing Engine Applications https://mathworks.com/help/matlab/matlab_external/engine-applications-overview.html Visited on 02.01.2022

[31] Matlab Documentation - Connect C++ to Running MATLAB Session https://www.mathworks.com/help/matlab/matlab_external/connect-to-running-matlab-session-1.html Visited on 02.01.2022

[32] Matlab Documentation - Start MATLAB Sessions from C++ https://www.mathworks.com/help/matlab/matlab_external/start-and-connect-to-a-matlab-sessions-from-c.html Visited on 02.01.2022

[33] Matlab Documentation - MATLAB Data API https://www.mathworks.com/help/matlab/matlab-data-array.html Visited on 02.01.2022

[34] Matlab Documentation - Introducing MATLAB Engine APIs for C and Fortran https://www.mathworks.com/help/matlab/matlab_external/introducing-matlab-engine.html Visited on 02.01.2022

[35] Matlab Documentation - C Matrix API https://www.mathworks.com/help/matlab/cc-mx-matrix-library.html Visited on 02.01.2022

[36] Matlab Documentation - .NET Libraries in MATLAB https://www.mathworks.com/help/matlab/using-net-libraries-in-matlab.html Visited on 02.01.2022

[37] Matlab Documentation - .Limitations to .NET Support https://www.mathworks.com/help/matlab/using-net-libraries-in-matlab.html Visited on 02.01.2022

[38] Matlab Documentation - TCP/IP Send https://www.mathworks.com/help/instrument/tcpipsend.html Visited on 02.01.2022

[39] Matlab Documentation - TCP/IP Receive https://www.mathworks.com/help/instrument/tcpipreceive.html Visited on 02.01.2022

[40] Matlab Documentation - MATLAB Runtime https://www.mathworks.com/products/compiler/matlab-runtime.html Visited on 02.01.2022

[41] Matlab Documentation - Funktionen https://www.mathworks.com/help/referencelist.html?type=function&capability=codegen&listtype=alpha Visited on 02.01.2022

[42] Matlab Documentation - MATLAB Language Features Supported for C/C++ Code Generation https://www.mathworks.com/help/coder/ug/matlab-language-features-supported-for-code-generation.html Visited on 02.01.2022

[43] Matlab Documentation - Compare System Target File Support Across Products https://www.mathworks.com/help/rtw/ug/compare-system-target-file-support.html Visited on 02.01.2022

[44] Matlab Documentation - Configure STF-Related Code Generation Parameters https://www.mathworks.com/help/ecoder/ug/configure-stf-related-code-generation-parameters.html#bu2qow8-2 Visited on 02.01.2022

[45] Matlab Documentation - Embedded Coder https://www.mathworks.com/products/embedded-coder.html Visited on 02.01.2022

[46] Matlab Documentation - Simulink Coder https://www.mathworks.com/products/simulink-coder.html Visited on 02.01.2022

[47] Matlab Documentation - Protect Models to Conceal Contents https://www.mathworks.com/help/ecoder/ug/create-a-protected-model-using-the-model-block-context-menu.html Visited on 02.01.2022

[48] Matlab Documentation - Generate Shared Library for Export to External Code Base https://www.mathworks.com/help/ecoder/ug/export-generated-shared-libraries.html Visited on 02.01.2022

[49] Functional Mock-up Interface for Model Exchange and Co-Simulation 2.0 https://fmi-standard.org/downloads/ 25.07.2014

[50] Matlab Documentation Answers - What's the difference between MATLAB function block and Interpreted MATLAB function block? https://www.mathworks.com/matlabcentral/answers/227446-what-s-the-difference-between-matlab-function-block-and-interpreted-matlab-function-block Visited on 02.01.2022

[51] Matlab Documentation - Implementing MATLAB Functions Using Blocks https://www.mathworks.com/help/simulink/ug/what-is-a-matlab-function-block.html Visited on 02.01.2022

[52] Matlab Documentation - C/C++ S-Function Basics https://de.mathworks.com/help/simulink/s-function-concepts-c.html?s_tid=CRUX_lftnav Visited on 02.01.2022

[53] Matlab Documentation - What Is an S-Function? https://de.mathworks.com/help/simulink/sfg/what-is-an-s-function.html Visited on 02.01.2022

[54] Matlab Documentation - Write Level-2 MATLAB S-Functions https://de.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html#f7-68222 Visited on 02.01.2022

[55] Matlab Documentation - S-Function Features and Limitations https://www.mathworks.com/help/simulink/sfg/s-function-features.html Visited on 02.01.2022

[56] Matlab Documentation - Create a Basic C MEX S-Function https://www.mathworks.com/help/simulink/sfg/example-of-a-basic-c-mex-s-function.html Visited on 02.01.2022

[57] Electrical and Computer Engineering Northwestern University - Writing S-Functions http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/simulink/sfg/sfun_c++2.html Visited on 02.01.2022

[58] Matlab Documentation - S-Function Builder https://www.mathworks.com/help/simulink/slref/sfunctionbuilder.html Visited on 02.01.2022

[59] Matlab Documentation - Use a Bus Signal with S-Function Builder to Create an S-Function https://www.mathworks.com/help/simulink/sfg/building-s-functions-automatically.html Visited on 02.01.2022

[60] Lawrence Berkeley National Laboratory - Building Controls Virtual Test Bed https://simulationresearch.lbl.gov/bcvtb Visited on 02.01.2022

[61] Matlab Documentation - WSDL (Web Services Description Language) https://www.mathworks.com/help/matlab/call-wsdl-web-services.html Visited on 02.01.2022

[62] Matlab Documentation - Comparison of Signal Loading Techniques https://www.mathworks.com/help/simulink/ug/comparison-of-signal-loading-techniques.html Visited on 02.01.2022

[63] Matlab Documentation - Overview of Signal Loading Techniques https://www.mathworks.com/help/simulink/ug/signal-loading-techniques.html Visited on 02.01.2022

[64] Matlab Documentation - From File https://de.mathworks.com/help/simulink/slref/fromfile.html Visited on 02.01.2022

[65] Matlab File Exchange - XML Parse-n-Find https://de.mathworks.com/help/simulink/slref/fromfile.html Visited on 02.01.2022

[66] Matlab Documentation - jsondecode https://www.mathworks.com/help/matlab/ref/jsondecode.html Visited on 02.01.2022

[67] Matlab File Exchange - JSONLab: a toolbox to encode/decode JSON files https://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab-a-toolbox-to-encode-decode-json-files Visited on 02.01.2022

[68] Matlab Documentation - coder.ceval https://www.mathworks.com/help/simulink/slref/coder.ceval.html Visited on 02.01.2022

[69] Matlab Documentation - Release Notes https://de.mathworks.com/help/simulink/release-notes.html Visited on 02.01.2022

[70] Krisper M., Iber J., Rauter T. and Kreiner C.: *Physical Quantity: Towards a Pattern Language for Quantities and Units in Physical Calculations*, Proceedings of Pattern Languages of Programs, EuroPLoP '17, pages 9:1–9:20, NY, USA. ACM, 2017

[71] Microsoft .NET Documentation - DllImportAttribute Class https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.dllimportattribute?view=net-5.0 Visited on 02.01.2022

[72] Matlab Documentation Answers - How do I debug and set breakpoints in custom C code used in a MATLAB Function block, Stateflow, or a C Caller block? https://de.mathworks.com/matlabcentral/answers/362169-how-do-i-debug-and-set-breakpoints-in-custom-c-code-used-in-a-matlab-function-block-stateflow-or-a Visited on 02.01.2022